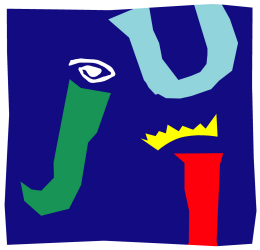

UNIVERSITAT JAUME·I

Doctoral School
Doctoral Programme in Computer Science

Game Development Based on Multi-agent Systems

Doctoral Thesis

Carlos Marín Lora

Supervisors: Dr. Miguel Chover Sellés (Universitat Jaume I)
Dr. José Martínez Sotoca (Universitat Jaume I)

CARLOS|
MARIN|
LORA

Firmado digitalmente por CARLOS|MARIN|LORA
Fecha: 2022.09.15 19:36:26 +02'00'

Firmado digitalmente por CHOVER SELLÉS, MIGUEL (FIRMA)
Fecha: 2022.09.15 19:10:55 +02'00'

Castellón (Spain)
September 2022


BY NC ND

The works composing this thesis were financed by:

- Project PID2019-106426RB-C32 funded by MCIN / AEI / 10.13039 / 501100011033 and ERDF "A way to make Europe".
- Project PDC2021-120997-C31 funded by MCIN / AEI / 10.13039 / 501100011033 and European Union "NextGenerationEU" / PRTR.
- Project RTI2018-098651-B-C54 funded by MCIN / AEI / 10.13039 / 501100011033.
- Projects UJI-B2018-56, UJI-2018-44 and UJI-FISABIO2020-04 funded by the Universitat Jaume I de Castellón.

Financiado por
la Unión Europea
NextGenerationEU

A Jose, prometo no despistarme

Thesis by Compendium of Publications:

- **Indexed Journals:**

  1. Marín-Lora, Carlos., Chover, Miguel., Sotoca, Jose M., & García, Luis. A. (2020). A Game Engine to Make Games as Multi-agent Systems. Advances in Engineering Software, 140, 102732. (Q1).
     `https://doi.org/10.1016/j.advengsoft.2019.102732`.
  2. Marín-Lora, Carlos., Sotoca, Jose. M., & Chover, Miguel. (2022, Apr). Improved Perception of Ceramic Molds Through Augmented Reality. Multimedia Tools and Applications. (Q2).
     `https://doi.org/10.1007/s11042-022-13168-5`.
  3. Chover, Miguel., Marín-Lora, Carlos., Rebollo, Cristina., & Remolar, Inmaculada. (2020). A Game Engine Designed to Simplify 2D Video Game Development. Multimedia Tools and Applications, volume 79, 12307–12328. (Q2).
     `https://doi.org/10.1007/s11042-019-08433-z`.
  4. Chover, Miguel., Sotoca, Jose M., & Marín-Lora, Carlos. (2022, May). Virtual Reality versus Desktop Experience in a Dangerous Goods Simulator. International Journal of Serious Games. (Q1).
     `https://doi.org/10.17083/ijsg.v9i2.493`.

- **International Conferences:**

  1. Marín-Lora, Carlos., Chover, Miguel., & Sotoca, Jose. M. (2020, April). A Game Logic Specification Proposal for 2D Video Games. In World Conference on Information Systems and Technologies (pp. 494-504).

Springer, Cham. (Core C).
https://doi.org/10.1007/978-3-030-45688-7_50.

2. Marín-Lora, Carlos., Cercós, Alejandro., Chover, Miguel., & Sotoca, Jose. M. (2020, April). A First Step to Specify Arcade Games as Multi-agent Systems. In World Conference on Information Systems and Technologies (pp. 369-379). Springer, Cham. (Core C).
https://doi.org/10.1007/978-3-030-45688-7_38.

3. Marín-Lora, Carlos., Chover, Miguel., & Sotoca, Jose M. (2021, October). A Multi-agent Specification for the Tetris Game. In International Symposium on Distributed Computing and Artificial Intelligence (pp. 169-178). Springer, Cham. (Core C).
https://doi.org/10.1007/978-3-030-86261-9_17.

This thesis has been accepted by the co-authors of the publications listed above that have waved the right to present them as a part of another PhD thesis

Acknowledgments

Carmen, Luismi and Cris, for teaching me to set the bar high and aim for the very best.

Rosa and Miguel, without you, I would not have been able to finish this thesis.

Lucía, for every moment that I have not been with you to write this thesis. For always being by my side and making me understand what is actually worthwhile.

Cooper, I couldn't have had a better partner while working on this.

Inés, for turning everything upside down and putting it all in the right perspective.

Cristina and Inma, for all your support, advice, and everything I have needed these years.

Jose, it seems impossible, but you taught me more over a coffee than in a classroom. Thank you for each "no, te equivocas" whatever the topic and its subsequent explanation.

Miguel, for being wrong about something you have been claiming and repeating for years: you could lift more than fifty kilos.

Resumen

En los últimos años, la popularidad de los videojuegos casuales ha atraído a perfiles creativos hacia el desarrollo de videojuegos. Un proceso multidisciplinar para el que se requieren conocimientos tanto técnicos como artísticos. A pesar de que existen herramientas como los motores de juegos que se conciben para facilitar la creación de videojuegos, estas siguen siendo aplicaciones complejas que precisan de experiencia técnica. Fundamentalmente, esto se debe a su origen como desarrollos de software o de programación tradicional, lo que supone una barrera de acceso para perfiles no técnicos. El objetivo de esta investigación pasa por proponer alternativas que faciliten el acceso a cualquier persona interesada, sin requerir conocimientos avanzados de programación, a la creación de videojuegos para dispositivos móviles, consolas, realidad virtual o realidad aumentada. Para ello, esta tesis trata de contribuir en el campo del desarrollo de videojuegos con tres contribuciones principales. En primer lugar con el diseño y desarrollo de un editor de videojuegos como herramienta de autor y de creación de contenidos, a partir de una especificación de juego simplificada que permite reducir la complejidad de la arquitectura del motor de juegos, y que introduce un entorno para la creación y edición de juegos fácil de usar. A continuación, se contribuye con la formalización teórica del motor de juegos utilizando la metodología de los sistemas multiagente y un método para definir la lógica de los juegos basado en una semántica de predicados, donde se cumple con los requisitos básicos de los sistemas multiagente, ajustando las características del motor de juegos sin afectar su potencial. Y por último, con el desarrollo de una metodología para la creación de juegos serios basada en sistemas multiagentes y con el estudio de la experiencia de juego en dos desarrollos de juegos serios como aplicaciones de realidad virtual y realidad aumentada.

Palabras clave: *Desarrollo de videojuegos*, *Editor de videojuegos*, *Motor de juegos*, *Lógica de juego*, *Sistemas multiagente*, *Juegos serios*.

Abstract

In the last few years, the popularity of casual video games has pushed creative profiles toward video game development, a multidisciplinary process that requires technical and artistic skills. Although there are tools such as game engines designed to facilitate access to the creation of video games, these are still complex applications that require technical expertise as a result of their origin as software developments and traditional programming, which is a barrier to access for non-technical profiles. This research aims to propose alternatives that ease access to any interested person, without requiring advanced programming knowledge, to the video games development for mobile devices, consoles, virtual reality, or augmented reality. For this purpose, this thesis tries to contribute to video game development with three main contributions. The first one deals with the design and development of a video game editor as an authoring and content creation tool based on a simplified game specification that reduces the complexity of the game engine architecture and introduces a user-friendly environment for game creation and editing. The second one is the theoretical formalization of the game engine using the multi-agent systems methodology, with a method to define the logic of the games based on predicate semantics while meeting the basic requirements of multi-agent systems and adjusting the features of the game engine without affecting its potential. And finally, the third one is the development of a methodology to create and specify serious games based on multi-agent systems, and with a game experience study brought from two serious games developments as virtual reality and augmented reality applications.

Keywords: *Video game development*, *Video game editor*, *Game engine*, *Game logic*, *Multi-agent system*, *Serious games*.

Index

List of Figures

List of Tables

List of Algorithms

Part I

Foreword

Chapter 1

Introduction

1.1 Context

Even though there is no generic definition for the concept of game, and the literature has a wide variety of proposals, it seems evident that it should express the idea of "*recreational activity controlled by rules*" [Juu10, Kos13]. Other authors also define the concept of video game as "a game that we play through an audiovisual device, and that can rely on a story" [Nic05]; as "*a puzzle-game, a toy-game or any type of game that runs on an audiovisual device*" [Geo75]; or as "*one or several series of challenges causally connected in a simulated environment*" [RA03]. The generic definition of the video game concept seems to lean towards digital game or electronic game. The meaning of game and video game emanates that a video game is a simulated virtual environment structured under rules.

In slang, the rule structures of a video game are known as game logic or game mechanics. At a conceptual level, the mechanics represent the actions that the player can perform in the game. The literature defines them as "*methods invoked by game elements, designed for interaction with the game state*" [Sic08], as "*any part of a game's rule system that covers a single possible type of interaction in the game*" [LB03], as "*descriptors of particular game components for data representation and algorithms*" or as "*actions, behaviors, and control mechanisms offered to the player within a game context*" [HLZ04]. However, a recent systematic review indicates a lack of consensus in the community on the formal definition [LTC21].

From the definitions found, three concepts seem to be fundamental: rules, interaction, and actions [Kos13]. Considering rules as the space of possibilities in which

actions are available, interaction as the methods by which the player communicates with the game, and actions as the changes produced in the state of the game. In a traditional game such as *Rock, Paper, Scissors*, the basic game mechanic is to reveal one's hand (interaction) in a particular disposition chosen from among three possibilities (rules), each of which is a winner over one of the others (actions). In a video game like *Ori and the Blind Forest*, the character unlocks different jump levels according to its progress in the game (rules) that the player can trigger with key combinations (interaction) and that allow access to new levels of the game map (actions). Or in a real-time strategy video game like *Age of Empires* the player can attack buildings or move units (actions) to a visible area of the map (rules) by selecting the units and clicking the target position with the computer pointer (interaction).

A critical aspect in the development of video games is the definition and implementation of game logic or game mechanics. The logic in a video game includes the methods and procedures that define the behaviors of the elements that are part of a game [Juu11]. Thankfully, and unlike artistic content, logic or mechanics are relatively exportable between games as it is usual to copy or adapt behaviors between games [Kos13].

During game development, logic-related tasks try to extract the specifications, requirements, and constraints established in the design and make them work in the video game context in real-time. That means transforming the design into code as any other software development, where the entities that perform the actions are known as game objects. These objects are the basis of video game development since they are the elements that make up the games and can represent characters, scenarios, sound sources, or managers, depending on the tasks or mechanics assigned to them. Conceptually, game objects are one of the fundamental data structures of video game development environments known as game engines.

A game engine is a framework or a set of tools that try to improve the development process of a video game. They aim to assist and integrate as many tools as possible to unify video game development tasks and to create reusable software that eases development and reduces production time/costs. In this way, developers spend less time on aspects that are not relevant to the overall idea of the game but are of vital importance to the resulting game experience.

Regardless of the degree of specialization of the game engine, the most common tasks they perform range from rendering for 2D and 3D graphics, through collision

detection and physics integration, sound reproduction, animations, user interaction, and logic evaluation via scripting, to memory management.

From an organizational standpoint, game design patterns and game engine architectures exploit features that are relevant to the project to be developed [Nys14]. For instance, there are standalone game engines with runtime architectures and hybrid editing and execution tools [Gre18]. This second option shares all data structures between all game engine components, avoiding having one version of the data structures for execution and another for edition. It speeds up the development process as it allows debugging the game directly in the editor.

These data structures, as previously mentioned, usually have the game object as their principal entity and represent any element in the game world that needs to be evaluated, updated, or rendered. Conceptually, they are descendants of the traditional objects of object-oriented programming [Gre18]. Each game engine establishes a relationship between them to organize their interactions, evaluation, and rendering using hierarchical structures such as scene graphs or sequential structures such as lists. The organizational structure of games is also essential for saving and loading games.

The way the engine runs the game is the game loop. A game loop is an iterative process that controls the overall game flow. The game is kept in continuous cyclic execution until it stops, either because the game ends or because the player exits. An example iteration of this loop in a generic engine would run through the set of game objects and for each of them to evaluate their physics and logic and then execute their audio and rendering components. At the end of one loop iteration, it would refresh the general properties of the active scene and the game, along with the variables associated with user interaction. In a traditional evaluation flow of information, a game engine evaluates the state of the game in a particular order. First, internal processes such as memory and resource management, user interaction management, physics or logic evaluation, and processes that generate game output such as rendering or audio are evaluated [Uni22].

Within this loop, the developer can include actions to define methods to complete specific tasks in the game or during engine execution, such as for debugging. Currently, the definition of these methods happens through scripting, either with a traditional programming language or through a visual programming system [Gre18]. This way of proceeding uses code as components that can be assigned to one or more game elements indistinctly, easing access to content creators and speeding

up game development. Among the processes and languages for which scripting is used, two categories stand out: data definition and real-time execution [Gre18]. The former generates simple data structures to support the definition and management of game logic. And the execution ones are usually based on simple languages, interpreted by a virtual machine, and encapsulated that encourage access to designers or users without programming knowledge.

As a summary, within a game engine, the developer must compose and organize the set of game objects according to the game design and define the mechanics or tasks they must perform to fulfill their design purposes. Usually, these tasks are implemented under a scripting system based on a programming language. The choice of a game engine, scripting system, and programming language depends on the context of the game design and the platform on which it will run. In today's industry, most games are developed on commercial game engines such as *Unity* and *Unreal Engine* or open-source engines such as *Godot*, and with programming languages such as *C++*, *C#*, *JavaScript*, or *Java* in generalist code editors such as *Visual Studio*, *Sublime Text*, and/or *JetBrains Rider*. An alternative to this process is visual programming languages such as *Unity Bolt* or *Unreal Blueprints*.

1.2 Motivation

The development of any video game is a multidisciplinary process that requires artistic, technical, conceptual design and project management skills, and where it is necessary to perform a series of more or less common general tasks regardless of the scope and purpose of the project. The differences between the task structure from one game to another vary in the conceptual shades of each project. The first task to be performed is the definition of the game structure. This structure should describe the types of game objects in the game, the environment controller and the game cycle, and the hierarchies of game objects that define the relationships between them. The second task would pass through assets needed to compose the game and their import into the game engine context. The last task would be to develop the set of game logic or mechanics that define the gameplay. These three tasks, with their variations, require previous background, which is an access barrier for users with creative profiles that do not exist in other visual arts like the cinema.

As the industry moved forward, and aiming to make content creation accessible to more people, game engines have been adding functionalities such as visual pro-

gramming systems for the mechanics' definition. In this way, developers can visually compose scenes, establish interaction, determine the behaviors of game characters, and even debug errors without relying on traditional programming. However, these alternatives usually encapsulate code functions in components that the user can arrange and associate interactively. This process bypasses code dependency but does not reduce the inherent complexity generated by the variety of functions that the development libraries have. Because the set of options they have continues to be very large and complex to handle for a user with no programming experience [AS10]. This problem becomes worse with the tendency to add more and more functions and more options to the engines. By making their use even more complicated and stretching the learning curves of these applications. The opposite road goes through knowing the essential requirements that a game needs to be executed.

Furthermore, some authors can also be found in the literature pointing out a lack of formalism in the video game development processes and other systematic problems such as the absence of a generic language explicitly oriented to game development [AEMC08]. One of the possible solutions is the multi-agent systems, which are often used as a design and organization pattern for software and applications with autonomous behaviors [Nys14]. The study of the organization, architecture, and characteristics of game engines suggests a close relationship between the concepts of game and multi-agent system. This arises from the comparison between game elements and the definition of agents, their relationships, and their communication protocols and cooperation mechanisms. Currently, a developer can use general methods to describe and specify video games, such as the Game Design Document (GDD) or the Game Description Language (GDL) [GLP05], to define the elements, functionalities, and interactions of game elements. However, they are usually oriented to be implemented in specific contexts and environments, which determines the game definition from the beginning to its implementation in the selected platform.

1.3 Goals

From the above, the work carried out during the development of this thesis proposes methods to improve the techniques of design, development, and specification of video games. For this purpose, three aspects are proposed: the design and development of a game engine, its theoretical formalization based on the methodology

of multi-agent systems, and a method for the specification of games as multi-agent systems.

In brief, the specific objectives of this research are as follows:

- Design and develop a game engine that reduces their use complexity without affecting its potential.
- Formalize the game engine to produce games as multi-agent systems.
- Define a method for specifying games as platform-independent multi-agent systems.
- Evaluate the gaming experience of games produced following the proposed specification model.

1.4 Previous Work

The work carried out before this thesis includes preliminary work on the relationship between multi-agent systems and game engines and the features of programming learning systems based on the creation of video games. Specifically, three papers published in national and international conferences can be highlighted.

The first one is titled "*Prototyping a Game Engine Architecture as a Multi-agent System*" and proposes the implementation of a game engine architecture as a multi-agent system, where the modules that compose the engine are agents that perform the necessary tasks to run an arcade game properly. The second one was published as "*Gamesonomy vs Scratch: Two Different Ways to Introduce Programming*", where the effectiveness of the educational method of teaching programming is analyzed by comparing two visual programming languages, one of them embedded within a game engine. Finally, the third one was titled as "*A Behaviour Specification System for Video Games Development*" and it presents a preliminary study on a game logic specification based on first-order logic, through which a developer can solve the mechanics of an arcade game. References to these three papers are listed below.

- *Marín-Lora, Carlos., Chover, Miguel., & Sotoca, Jose M. (2019). Prototyping a Game Engine Architecture as a Multi-agent System. In 27th International Conference in Central Europe on Computer Graphics, Visualization and Computer Vision (WSCG 2019). (Core B).*
- *Marín-Lora, Carlos., Chover, Miguel., & Sotoca, Jose M. (2019). A Behaviour Specification System for Video Games Development. In Spanish Computer Graphics Conference (CEIG 2019).*

- *Rebollo, Cristina., Marín-Lora, Carlos., Remolar, Inmaculada., Chover, Miguel. (2018). Gamesonomy vs Scratch: Two Different Ways to Introduce Programming. In 15th International Conference On Cognition And Exploratory Learning In The Digital Age (CELDA 2018). Ed. IADIS Press. (Core C).*

1.5 Contributions

This thesis describes the development of technology to support the creation of video games. The work begins with the study, design, and implementation of a game engine, continues with the definition of a methodology for the specification of games as multi-agent systems and ends by applying the specification methodology for the development of serious games. More specifically, and narrowly speaking about the papers that structure this thesis by a compendium of articles, the thesis is composed of seven papers categorized into three main contributions.

The first contribution consists of a paper published in an international journal titled "*A Game Engine Designed to Simplify 2D Video Game Development*" where is proposed the design and development of a video game editor as an authoring and content creation tool based on a simplified game specification that reduces the complexity of the game engine architecture and introduces an environment for easy-to-use game creation and editing.

The second contribution begins with the formalization of the game engine using the methodology of multi-agent systems and with a proposal to define the logic of the games based on predicate semantics, where the basic requirements of multi-agent systems are met, adjusting the characteristics of the game engine without affecting its potential. This contribution relies on four publications, one international journal, and three international conferences. The first one is titled "*A Game Engine to Make Games as Multi-agent Systems*", while the remaining three are composed of adjacent papers where the multi-agent methodology was applied to the specification of games and the specification of behaviors in different genres and platforms. These papers were published with the titles "*A Game Logic Specification Proposal for 2D Video Games*", "*A First Step to Specify Arcade Games as Multi-agent Systems*" and "*A Multi-agent Specification for the Tetris Game*".

Finally, the third contribution develops a methodology for the specification and implementation of games as multi-agent systems on any platform by including the development of two serious games as virtual and augmented reality applications as multi-agent systems and the study of the resulting game experience. These

two applications form the two publications in international journals that make up this latest contribution. The first was titled "*Improved Perception of Ceramic Molds Through Augmented Reality*" and the second was titled "*Virtual Reality versus Desktop Experience in a Dangerous Goods Simulator*".

Below there is a brief summary of each of these papers with a short review of the contributions made in the order they appear in this section.

1.5.1 Design and Development of a Game Engine

A Game Engine Designed to Simplify 2D Video Game Development

- *Chover, Miguel., Marín-Lora, Carlos., Rebollo, Cristina., & Remolar, Inmaculada. (2020). A Game Engine Designed to Simplify 2D Video Game Development. Multimedia Tools and Applications, volume 79, 12307–12328. (Q2).*
  `https://doi.org/10.1007/s11042-019-08433-z`.

The growing popularity of casual games in recent years has promoted the development of new tools to ease the creation of video games. This process advances their democratization so that anyone interested can develop them without programming knowledge. However, most game development environments are based on the traditional way of programming and require advanced technical knowledge. This paper presents a new 2D game engine in which, based on a simplified game specification, the complexity of the game engine architecture is reduced and easy-to-use game creation and editing environments are introduced. The goal is to reduce the complexity of video game development processes and facilitate access to development for people with non-technical profiles. For this purpose, a new approach to developing these interactive applications is proposed compared to the traditional programming method. Instead of using the data structures of a programming language, a reduced data model is proposed, with a list of predefined components and parameters and a restricted set of actions and conditions to define game behaviors, without the need to use complex data structures. The validation of its operation passes through arcade game development experiments with inexperienced users. The results show that inexperienced users can successfully develop arcade games by using this game engine. The contribution focuses on the implementation of the game engine based on a data model concept that had already been thought out by the co-authors of the paper. In addition to the design, management, and analysis of the experiments performed.

1.5.2 Game Engine Formalization Using Multi-agent Systems

A Game Engine to Make Games as Multi-agent Systems

- *Marín-Lora, Carlos., Chover, Miguel., Sotoca, Jose M., & García, Luis. A. (2020). A Game Engine to Make Games as Multi-agent Systems. Advances in Engineering Software, 140, 102732. (Q1).* `https://doi.org/10.1016/j.advengsoft.2019.102732`.

Video games are applications that feature design patterns that resemble multi-agent systems. Where the elements that compose the games are like autonomous agents that interact with each other to describe complex systems within a shared environment or social space. This paper presents the definition of a game engine that can produce games that meet the requirements of a multi-agent system following the formal definition of multi-agent systems to perform a description of the game and its essential elements. The purpose is to develop a game engine that can generate games as multi-agent systems. In other words, games that meet the basic requirements of multi-agent systems by adjusting the system features without affecting their performance and potential. Where the actors or agents of the game engine have a set of properties and behavioral rules with which to interact with the game environment. For this interaction, the actors have a behavior definition system established through formal semantics based on predicate logic. The paper also includes the specification of a set of games as use cases for their development as multi-agent systems to validate their operation and possibilities. These use cases show that the system fulfills its task, and it is observed that it can generate complex behaviors from reduced logical semantics as behavior definition descriptors. The contribution made in this work deals with the study of multi-agent systems, the definition of the formal analogy between video games and multi-agent systems, and the implementation of the engine and the use cases.

A Game Logic Specification Proposal for 2D Video Games

- *Marín-Lora, Carlos., Chover, Miguel., & Sotoca, Jose. M. (2020, April). A Game Logic Specification Proposal for 2D Video Games. In World Conference on Information Systems and Technologies (pp. 494-504). Springer, Cham. (Core C).* `https://doi.org/10.1007/978-3-030-45688-7_50`.

Game engines are essential applications in game development, given that they are designed to assist in the creation of game content. However, the definition

and specification of game logic remain complex. The mechanisms involved in the behavior definition are not easy to standardize, given their dependence on the subjective design of the game logic and the features of the selected programming language. In this sense, this paper proposes a game logic specification for 2D video games based on a reduced set of behavioral elements. This proposal relies on the study and analysis of the literature on behavior and game logic definition. The analysis performed on the requirements and the characteristics of existing game engines indicates that there is room for improvement in determining the basic requirements of games and the correct procedures for their design. With the information gathered, semantics based on first-order logic have been defined to generate rules from a game data model and a reduced set of actions and conditions. All this without hierarchies, loops, or complex data structures such as matrices or vectors. The proposed model has been tested with several games and against two other commercial game engines to determine its validity and potential. The first one is with behavior definition tools based on traditional scripts and the second one relies on message passing. The methodology followed for this comparison consisted of the game description with the proposed semantics as a reference and their subsequent implementation in the other game logic systems by several users. The results show that the users perceived the system as more usable than the other commercial systems analyzed. The contribution made in this research involves the study of behavior specification systems for autonomous systems and knowledge representation systems, the definition of a logic specification system for 2D games, and its subsequent integration into the multi-agent game engine.

A First Step to Specify Arcade Games as Multi-agent Systems

- *Marín-Lora, Carlos., Cercós, Alejandro., Chover, Miguel., & Sotoca, Jose. M. (2020, April). A First Step to Specify Arcade Games as Multi-agent Systems. In World Conference on Information Systems and Technologies (pp. 369-379). Springer, Cham. (Core C).* `https://doi.org/10.1007/978-3-030-45688-7_38`.

The lack of formalism in video game development is an obstacle to incorporating new professionals in this field. Even though there are general proposals to describe and specify video games with techniques such as the Game Design Document (GDD) or the Game Development Language (GDL), they are focused on specific implementations. This paper proposes a method for the definition, specification, and implementation of a video game based on multi-agent systems, where its elements,

functionalities, and interactions are established independently of the platform used for its development. The goal is to demonstrate that video games and multi-agent systems share several features and that it is possible to improve video game development processes in this way. As a demonstrator, the classic arcade game *Frogger* has been used to test its validity and capabilities. This game has been defined in its general form and implemented on three different platforms following the same specification. Each of these implementations has been done using different engines, languages, and programming techniques, but in any case, meeting the requirements of the game and the multi-agent systems. The study suggests that the features of multi-agent systems fit as a starting point for the definition of video games and asks to keep working on methods that foster a symbiosis between these concepts. In addition, the incorporation of agent specification systems in the development of video games can ease their comprehension before implementation, which could facilitate access to the sector of professionals who do not have technical experience in the creation of video games. The contribution in this work focuses on the specification of the game for its implementation in the three platforms, the supervision of the implementation of the games, and the analysis of the differences between their implementations.

A Multi-agent Specification for the Tetris Game

- *Marín-Lora, Carlos., Chover, Miguel., & Sotoca, Jose M. (2021, October). A Multi-agent Specification for the Tetris Game. In International Symposium on Distributed Computing and Artificial Intelligence (pp. 169-178). Springer, Cham. (Core C).*
  `https://doi.org/10.1007/978-3-030-86261-9_17`.

In the video game development industry, tasks related to design and specification require support to translate game features into implementations. These support systems must clearly define the elements, functionalities, and interactions of the game elements, and they must also be established independently of the target platform for development. From a study for the specification of games as multi-agent systems, this paper attempts to test if the results of that study are transferable platform-independent. This study leans on a game engine created to generate games as multi-agent systems. The objective is to validate the hypothesis that a game can be defined and specified as a system of interacting agents and that it is possible to implement it on different platforms. As a case study, the classic game Tetris has been used, a game whose nature suggests that its implementation should be

composed of vector and matrix data structures. The game has been defined and specified according to a reference model with three different types of agents. This specification has been implemented on three different platforms with satisfactory results, consequently validating the starting hypothesis. The contribution made in this work includes the analysis of the game requirements, the specification of the game as a multi-agent system, and its implementation.

1.5.3 Serious Games Development

Improved Perception of Ceramic Molds Through Augmented Reality

- *Marín-Lora, Carlos., Sotoca, Jose. M., & Chover, Miguel. (2022, Apr). Improved Perception of Ceramic Molds Through Augmented Reality. Multimedia Tools and Applications. (Q2).* `https://doi.org/10.1007/s11042-022-13168-5`.

Augmented reality is a technology that adds graphical information to the real world that the user is visualizing through a device. The development of applications based on this technology requires knowledge about how some parameters, such as the user's physical presence, affect the perception and evaluation of the experience. This paper presents an augmented reality application for the presentation and marketing of ceramic molds. The application combines a physical ceramic mold with virtual components of the mold's interior in real-time, where the sequence of actions goes through an avatar that explains the details of its operation, and where users visualize the graphic information of the mold, placed between them and a large format television screen as if it was a mirror. The study derived from this application puts forward a theoretical framework that seeks to understand how users perceive and evaluate the benefits and quality of the experience through their physical presence, compared to viewing the same experience through a video. The experiment results showed that the integration of the product into the environment and the users spatial presence had a positive effect on the perceived value in terms of usefulness and enjoyment, improved comfort in the purchase decision, and reinforced the overall opinion of the product. Confirming that content created for a clear and persuasive use should be remarkable for companies when promoting their products. The contribution made in this work consists of the definition of the experience requirements, its specification and implementation in a commercial game engine as a serious 3D game, the implementation of the application in context, and the analysis of the results of the participants' experience.

Virtual Reality versus Desktop Experience in a Dangerous Goods Simulator

- *Chover, Miguel., Sotoca, Jose M., & Marín-Lora, Carlos. (2022, May). Virtual Reality versus Desktop Experience in a Dangerous Goods Simulator. International Journal of Serious Games, 9(2), 63–77. (Q1).*
  `https://doi.org/10.17083/ijsg.v9i2.493`.

Virtual reality applications have become a trend in training simulators as an alternative to desktop applications as they enhance the reality experience felt by the user, including tactile, visual, and acoustic sensations. This is a breakthrough in terms of interaction and total immersion of the player in the game. However, studies about how virtual reality affects gameplay are still limited, and it is not clear how interaction through virtual reality controllers can help or harm the player's experience. In this regard, this paper analyzes the differences between playing a serious first-person game on a desktop computer versus playing in virtual reality. The game was implemented in two versions of a dangerous goods unloading simulator. The first was developed as a classic desktop game with keyboard and mouse-based interaction, while the second was for virtual reality devices. The evaluation of the user experience relied on the in-game version of the Game Experience Questionnaire. With this, aspects related to immersion, flow, positive emotions, and psychological needs were compared for these two platforms. The study shows that the virtual reality experience produces a better overall gaming experience for most of the items analyzed. However, the results show a significant dependency between the type of application and the gaming experience induced in the player. The contribution made in this work involves the specification of the serious game and the analysis of the data obtained from the game experience questionnaire.

1.6 Outline

This document has been organized into five blocks to structure and group the contents and contributions of this thesis. The first introduces a foreword about the thesis's context, motivation, and contributions with Chapter 1. The second one deals with the design and development of a game engine and consists of Chapter 2 of the thesis introducing the paper "*A Game Engine Designed to Simplify 2D Video Game Development*". The third block presents the formalization of the game engine using the multi-agent systems methodology in the main paper and includes three additional works carried out in parallel to define and specify games and their logic.

These works are divided into four chapters, starting with Chapter 3 titled as "*A Game Engine to Make Games as Multi-agent Systems*" and continuing with Chapters 4, 5 and 6 presenting the papers "*A Game Logic Specification Proposal for 2D Video Games*", "*A First Step to Specify Arcade Games as Multi-agent Systems*" and "*A Multi-agent Specification for the Tetris Game*", respectively. The fourth block introduces the development of two virtual and augmented reality applications following the model proposed in the second block. This section consists of two chapters: Chapter 7 with the paper "*Improved Perception of Ceramic Molds Through Augmented Reality*" and Chapter 8 including the paper "*Virtual Reality versus Desktop Experience in a Dangerous Goods Simulator*". Finally, the fifth block provides an afterword with the conclusions as well as the proposed lines for future work with the Chapter 9.

Part II

Design and Development of a Game Engine

Chapter 2

A Game Engine Designed to Simplify 2D Video Game Development

Publication

Chover, Miguel., Marín-Lora, Carlos., Rebollo, Cristina., & Remolar, Inmaculada. (2020). A Game Engine Designed to Simplify 2D Video Game Development. Multimedia Tools and Applications, volume 79, 12307–12328. (Q2).
https://doi.org/10.1007/s11042-019-08433-z.

Abstract

In recent years, the increasing popularity of casual games for mobile and web has promoted the development of new editors to make video games easier to create. The development of these interactive applications is on its way to becoming democratized, so that anyone interested, without any advanced knowledge of programming, can create them for devices such as mobile phones or consoles. Nevertheless, most game development environments rely on the traditional way of programming and need advanced technical skills, even despite today's improvements. This paper presents a new 2D game engine that reduces the complexity of video game development processes. The game specification has been simplified, decreasing the complexity of the engine architecture and introducing a very easy-to-use editing environment for game creation. The engine presented here allows the behavior of the game objects to be defined using a small set of conditions and actions without

the need to use complex data structures. Some experiments have been designed to validate its ease of use and its capacity for the creation of a wide variety of games. To test it, users with little experience in programming have developed arcade games using the presented environment as proof of its ease concerning other comparable software. Results obtained endorse the concept and the hypothesis of its easiness of use and demonstrate the engine's potential.

Keywords

Game development, Game engine, Game editor

2.1 Introduction

Video game creation is a very complex process where the participation of a multidisciplinary team is required, as well as the use of tools to assist the production of content. Game developers not only have to create games interesting enough to captivate players, but they also have to face the complex technical features required for today's computer games: graphical resources, interaction mechanisms and behavior definition [Gre18]. To simplify the problem, game development has evolved quickly since the mid-1990s, mainly because of the emergence of game engines. These tools aim to create reusable software to provide an easier way to generate games and reduce their production times. Although in their early stages the primary concern of these engines was the rendering system, other fields such as artificial intelligence, animation, physics, sound, or networking were added over time.

Over the years, and to make content creation accessible to more people, some game engines have been incorporating visual programming systems [BM19, KHA97] for the definition of game behavior. In this way, the developers are now able to visually compose the scenes, set the interaction, determine the behaviors of the game characters, and even debug errors. However, even though since its conception game engines have simplified the creation of video games, these tools are still too complicated to use. The set of functions and elements they have is still very large and is still difficult to handle for a non-programmer user.

As a starting point, the democratization of game development seems to be easier to achieve in 2D. As an example, one of the most popular environments for creating interactive 2D content through visual programming is called Scratch [MRR+10]. In

this case, traditional programming is omitted by encapsulating code functions in block-shaped nodes that the user has to organize to create their algorithms. However, in this way, it does not eliminate the inherent complexity of traditional programming methodologies and although the environment is more attractive, complexity is similar to traditional programming. For these reasons, the development of tools that facilitate and make accessible the creation of video games for everyone remains an open problem and has a great interest.

This paper presents a Simplified Game Engine (SGE) designed to ease the game development process by providing tools oriented toward non-programmers. This 2D game engine has been designed by simplifying each user-dependent process as much as possible to provide a most satisfying level of abstraction in terms of technology. The contributions of the work are aimed at the simplification of the video game specification, emphasizing the simplification of the game logic definition, and can be summarized in the following ones:

- Simplification of the data model used to define games and a consequent specification of a simplified game engine architecture and editor.
- Elimination of hierarchical structures of game objects, common in most game engines to exploit the agent-based programming paradigm [WJ95].
- Creation of a visual programming system using binary decision trees, used in several fields beyond computer science [KAT+16, YPY+19].
- The engine does not include complex data structures such as vectors or matrices [DeL00], common in other visual systems such as Scratch. Elimination of repetition statements, since the iteration is provided by the game loop itself.

Finally, to validate the capabilities and ease of use of the engine, two experiments have been carried out with children without great programming knowledge. The main objective is to compare SGE with Scratch, one of the most widespread visual programming environments. Scratch is used to program scripts in some game engines such as Stencyl [LLW+14, Ste19].

The rest of the paper is organized as follows. In Section 2.2, the leading work in the area with state-of-the-art game engines and visual tools to learn to program is introduced. Next, in Section 2.3, the technical conception of this work is developed,

focusing on the game engine architecture and game specification along with the game editor and its behavior specification system. Thereafter, a complete use case example is presented comparing the programming using SGE and Scratch in Section 2.4. Next, in Section 2.5, an experiment carried out with children is explained and its results are presented and discussed in Sections 2.6 and 2.7, respectively. Finally, Section 2.8 concludes the main contributions of the proposal and its limitations with an outline of the ongoing work.

2.2 State of the Art

The video games industry, like any other, tries to minimize production costs to maximize profits [Fol07, Wil02]. During the mid-90s, some companies, such as IdSoftware, added modularization to their main engines intending to reuse the software. They developed the First Person Shooter (FPS) game Doom, from where any further addition or change was easy to implement by modifying levels, characters, weapons or even creating new games. This led to the game engine concept and provided tools to develop new games more easily. In the late 90s, some games, such as Id Software's Quake III and Epic Games' Unreal, were built on a modular and reusable conception. In this way, game engines improved the customization possibilities by adding coding features, for instance, scripting functionalities such as Quake C. From this point on, game development companies became aware of the commercial interest in game engine licenses and started looking for an additional source of income.

As time passes, the improvements in graphics hardware, visualization technology, and data structure are closing the gap between game engines for varying purposes. Today, it is possible to create 2D or 3D games with the same game engine. Even though specialization is still capital [Gre18], creating a Massive Multiplayer Online Game (MMOG) is quite different from an FPS. The required features can be very different for each game genre. For instance, 2D animated sprites are pretty simpler to set up than realistic 3D visualization algorithms [Kir04] or, in the same way, collision computations and physics simulations are far more complex in 3D [Mil10]. An example of these game engines is Unity [MW12, MSK15, Uni22], a mighty platform to develop 2D or 3D games where deep knowledge about game engine concepts, features, and advanced experience in the programming language C# is required to develop anything. Despite this, it is easy to perceive a trend toward simplification:

a 3D engine of the highest level such as Unreal Engine [Epi22] includes a visual programming system called Blueprints Visual Scripting [Val15] based on message passing, where programming is done by connecting game objects' components and functions. However, this engine has a high-level commercial purpose, so its use is still quite complex for non-technical people.

In response to the needs of these potential users, some companies have developed 2D game engines intended for creators without advanced programming knowledge. Its systems have visual editors to configure scenes, characters, and even gameplay mechanics without writing a line of code. Most of them include visual programming methods, an approach that can bring simplicity to this process through one of its visual scripting methodologies: block-like, flowcharts-inspired, dataflow or message-passing programming, finite-states machines, event-based rules, or behavior trees [BM19, KHA97].

Table 2.1 presents a summary of the analysis of some of the current game engines that allow the creation of 2D games. The table details the platform where they are executed, the scripting system they use, their visual programming methodology, and the number of functions or behavior descriptors each one has to configure those gameplay mechanics. The elements of the table are arranged in ascending hierarchy based on the number of functions or behavior descriptors and their ease of use.

The table begins with Flowlab [Flo19], which has fifty-three different elements to configure behaviors, and ends with Unity, where the action is conducted by hand-made C# scripts based on the complete language and some specific programming

Table 2.1: Classification for state-of-the-art 2D game engines

Game Engine	Platform	Scripting Method	Behaviour Specification	Elements *
Flowlab	Web	Visual Scripting	Message Passing	53
Game Salad	Desktop	Visual Scripting	Components	63
GDevelop	Desktop	Visual Scripting	Event System	106
Game Maker	Desktop	Game Maker Language	Message Passing	133
Construct 2	Desktop	JavaScript	Blocks	204
Stencyl	Desktop	Scratch	Blocks	C.L.
RPG Maker	Desktop	Ruby, C++, Java, JavaScript	Scripting	C.L.
Unreal	Desktop	C++, Visual Scripting	Message Passing	C.L.
Unity	Desktop	C#	Scripting	C.L.

* Number of predefined set of functions or behaviour descriptors | C.L. Complete Language

libraries and APIs. Many of them, like Game Maker [YoY19] or RPG Maker [RPG19], have systems based on lighter scripting tools, which keeps a dependency on code. Additionally, others such as Construct 2 [SL17] or Stencyl [LLW+14, Ste19] rely on block-like interfaces, this latter case working with Scratch [MRR+10], a visual programming concept developed by the Massachusetts Institute of Technology (MIT). Usually, these block-like systems are based on events GDevelop [Cor15] attaches its behavior definition system directly to events into a cross-platform engine with a visual programming interface. Finally, GameSalad [DX12, RRD12] presents a graphic interface that allows one to visually configure functioning by arranging and connecting components.

Even though these efforts are very significant steps forward toward game development complexity reduction, the use of this kind of software still requires high technical profiles and specific training, thereby excluding most of the potential users of these technologies [VMB+13, TZ05]. One of the main reasons is related to the transition process between traditional programming and visual programming: it has generally been done by transforming programming functions into components, which avoids dependence on the code but maintains the huge variety of functions of traditional APIs. This way of proceeding inherits a systematic problem from the development of game engines: there is no generic game engine language, there is deep darkness about the essential requirements that a game needs to be executed and the limits between games, genres, and engines of games are blurred [AS10, AEMC08]. With this in mind, it is necessary to find a simpler way to create games, where inexperienced users could develop their games by making use of graphical environments that do not require programming skills [GFZI12].

In the current literature, some works point out how complex can be for a beginner the approach to a problem through computational techniques [Cha05, MR02, RRR03] and the assistance that visual programming can provide [Bla96, Cha16]. At the educational level, different methodologies have been studied to introduce programming concepts, both with traditional coding [KLM14] and with visual programming [PGC+06]. It seems evident that visual programming can be an essential tool on the way toward the democratization of game development. Some authors have carried out experiences with students associating visual programming and computer games. For instance, some authors [OKD+15] present a study conducted on programming students to test the learning of basic programming concepts by creating games with Scratch [MRR+10], and others [CC07] display a study to eval-

uate an object-oriented programming learning methodology through videogames programming.

At a more specific level, some works have proposed models that combine visual programming methodologies with the elements that a game engine requires to define the behaviors of a game. In this line, Furtado et al. [FSRdA11] propose a description of game engines based on a more abstract and expressive set of layers. Furthermore, Zarraonandia et al. [ZDA17, ZDAR15] presented a conceptual model to organize the game features in a modular way, where the description and the definition required to create a combination of sub-games are based on a set of configurable elements and a basic vocabulary for each feature. Additionally, some software engineering methods have appeared as a possible plan to address this issue, proposing a systematization of the game development process [AS10, FS06, RC08]. All this analysis demonstrates that the creation of video games can be simplified and the development of new visual tools can make the creation of such content accessible to a large number of users.

2.3 The Simplified Game Engine

The main hypothesis of this work is to demonstrate that the complexity of a game engine can be reduced concerning the software architecture, the specification of the games, and the editor itself while maximizing its potential in terms of creating different types of arcade games. In this section, the architecture of the proposed system is introduced below. This architecture has been designed to be able to create a wide variety of games, including physical games. Subsequently, the specification of the games is described. Each game can be composed of different scenes, with actors that can have different behavior rules or scripts. All actors have the same properties which simplify both the specification of the games and the implementation of the engine. Finally, the game editor designed using the aesthetics of Google Material Design [Goo19] is briefly described.

2.3.1 The Game Engine Architecture

Essentially, most of the existing game engines have quite similar architectures and subsystems. This is because these modules are necessary for designing practically any game. A generalist system needs some modules that manage rendering, sound,

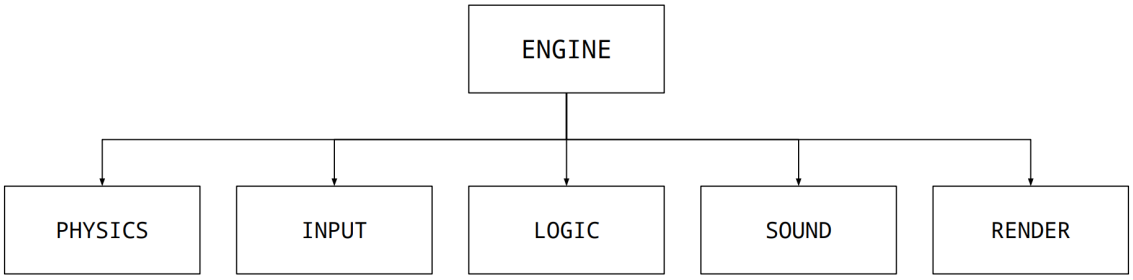

Figure 2.1: Classic game engine architecture

logic evaluation, input system, and physics. It is for this reason that this proposal has included a set of five modules: physical computing controllers, event management, logical evaluations, sound reproduction, and scene representation. A representation of this architecture can be seen in Figure 2.1.

The most important module that has been developed is the game logic module. For the rest of the modules, a high-level layer has been created that allows different libraries to be incorporated. For example, about the physics engine, both Box2D [Cat11] and Matter [Bru14] have been tested. Concerning the render engine, a 2D render has been developed from scratch and Pixi [VdS15, Goo13] has also been tested. With the game logic module, a system has been implemented that allows building the behavior rules using decision trees and a small set of conditions and actions (see section 2.3.2).

The Game Loop processes every action or condition in these five submodules on each iteration. It starts from the physics, goes through the event handling, continues with the game logic evaluation, and, finally, it represents the state of the Game by the sound and rendering modules. The implementation of a Game Loop in a classic game engine may become a very challenging and very complex task. According to Deloura [DeL00], this implementation usually absorbs the users in customized scripts for each game object to be executed in different stages of the Game, or even a different number of times per frame, making it necessary for the user to have a vast knowledge of the game engine structure and operation. This process has been considerably simplified in SGE, the logic of the game is always computed after the physics and input evaluation in each game cycle. This simplification can be done since most 2D games do not have great performance needs.

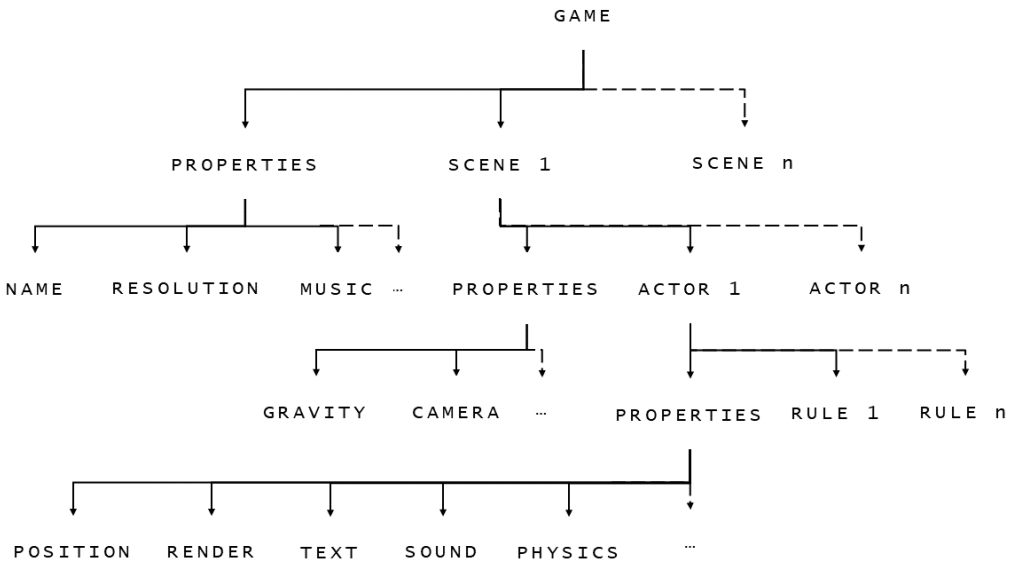

Figure 2.2: Game data structure

2.3.2 The Game Specification

An SGE Game can be represented as a set of Scenes with a list of Actors, where the Actors are the elements of the game and the Scene is the stage used by the Actors. The concept behind this setup is to make Actors work as independent agents [Bis08, WJ95] carrying out their roles and interacting both with each other and with the Scene configuration. Accordingly, a game specification has been developed, based on the Actor, and through its Properties and Rules, any of these elements can be configured or modified. A diagram of this configuration is presented in Figure 2.2. The system has no hierarchy of actors like most 3D game engines, which simplifies the game engine architecture. This absence of hierarchy has not been a problem to implement any mechanics in the arcade games that have been created so far.

The Game is the central object of the system's data structure. Its properties are mainly related to the resolution and the orientation of the screen along with the sound linked to the entire game. The user also can store new custom variables to meet some game requirements, for example, to store the total score that players reach. In turn, the Game is composed of Scenes. These are responsible for storing camera properties, such as position, angle, zoom, or gravity. The Actors, considered

the main component of the game, are assigned in the Scenes. This is the most complex component since it is the cornerstone on which all interaction rests. They also have their properties, related to their position, render, text, sound, and physical properties. In the same way as the other components of the game, it can store custom variables to help the development of the game. In addition, they have a list of rules assigned to them that perform the gameplay by combining actions included in some programmable nodes. As they are considered the main structure of the SGE, they are explained in detail in the following.

Actor Properties

The Actors are the only element in the Scene and they are organized by an ordered list to determine their order of viewing. All the elements arranged in a Scene, such as characters, backgrounds, decorations, sound emitters, markers, or HUDs, are Actors. They play a key role in the definition of the game. In a conventional game engine, the Actor concept would be expressed as game objects with different types of roles: lights, camera, sound, geometry, etc. However, this specification only requires the use of Actors, removing the reliance on game hierarchies, a central topic in traditional games implementation but, in fact, not essential to create most the classic arcade games.

The Actors have a predetermined set of Properties by default, some of them related to the visual appearance: color, line width, etc. Depending on their function in the game, they can be visible in the Scene, for example, when an Actor is assigned an image, it will be visible just as the gameplay is designed using the Actor's Properties and its Rules require. Besides images, they can represent other features in the game, such as sounds, text, numbers, or booleans.

Finally, the established set of Actor's properties can be classified and arranged in categories, to provide a better understanding for the final user. These properties are as follows:

- **General**. These properties deal with the Actor's position, scale, and rotation. They also include information about the collision shape profiling and if the object is enabled or disabled.

- **Graphic**. Related to visual and graphical properties: the Actor image and other transform options such as flip, repeat, and displacement.

- **Text**. The Actors can show text in a certain position with a particular font and style.

- **Sound**. A sound can be associated with the Actor. It presents some properties to allow the modulation of its volume, and some options to determine the moment it starts and if it is in a loop.

- **Physics**. These properties are related to the type of physics body: dynamic, kinematic, or static. Other characteristics such as the velocity and the properties that depend on the material, such as density or friction, are also included in this category.

- **Custom**. Additionally, it is possible to store information on variables to customize the games even further.

These properties can be modified from the behaviour rules of the actor itself or other actors of the same scene.

Actor Rules

Actors are in charge of managing every event that takes place in the Game. For this purpose, a rule system has been devised in order to define the logic and the interactive behaviours of the Game. A behaviour rule is determined by a decision tree system [Mil19] driven by a reduced set of Actions and Conditions, ready to be executed if the flow passes through them according to whether the Conditions are met or are not. An example of a rule is shown in Figure 2.3.

The Rule structure is defined by two elements: Actions and Conditions arranged in a decision tree. Both Actions and Conditions are prepared to work with arithmetical expressions and with mathematical functions: sin, cos, tan, asin, acos, atan, sqrt, random, etc.; and the data types supported by this system are numbers and booleans.

Actions are the defining elements of the specific behaviours of the Actors. A list of predefined actions is available to the game designer. In this sense, a thorough review has been carried out to generate a stable simplification in order to arrange a minimum set of predetermined Actions and thus to simplify the game logic. The study has resulted in fifteen Actions, some of which are described below:

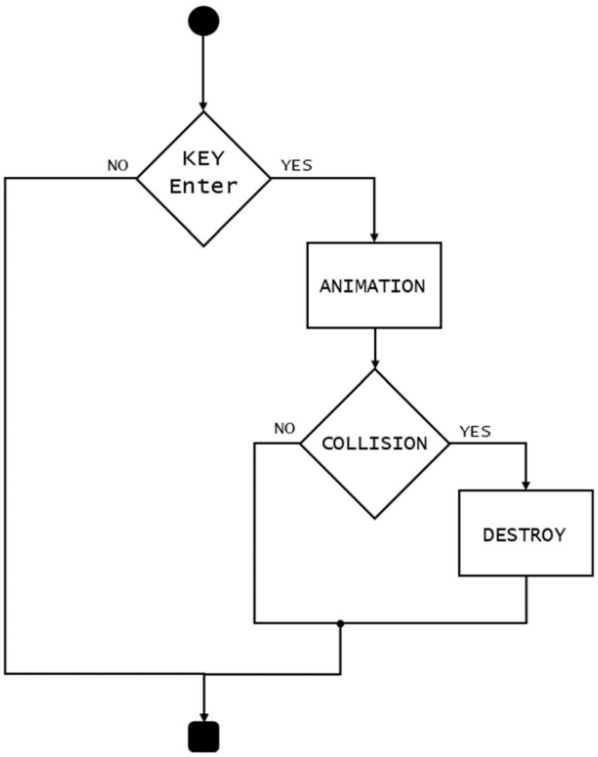

Figure 2.3: Diagram of a decision tree

- **Edit**. To change every Property by a specific value or by the result of an arithmetical expression. It works equally for the Game, the Scene or the Actor.
- **Animate**. To execute animations by arranging sprites on a specific frames-per-second rate.
- **Move / Move To**. To move the Actor a certain number of units on screen. It has a related Action called Move To, which causes it to travel towards a specific position or Actor.
- **Destroy**. This Action deletes the Actor from the Scene when is triggered.
- **Push / Push To / Torque**. To apply forces to the Actor. It also has a related Action called Push To and another pack called Torque that works with the same concept to apply angular forces.
- **Rotate / Rotate To**. To rotate the Actor a certain number of degrees. There is also Rotate to Action which allows rotating until reaching an angle.

- **Add Scene / Remove Scene / Go To Scene**. To handle the management of Scenes with Add Scene, Remove Scene and Go to Scene.
- **Spawn**. An automatic Actor copy generator.
- **Sound**. To start a sound by Play sound.

The conditions allow defining the execution flow for the actions. The scripting system includes the following types of conditions:

- **Compare**. This compares data values or expressions from any Game, Scene or Actor Properties with another value or expression.
- **Check**. To check if a boolean Property is met or it is not.
- **Collision**. This checks whether two Actors are colliding. It relies on the Actor's collision shape.
- **Keyboard**. To capture which key has been pressed on the keyboard.
- **Touch**. To manage the user interaction with mouse or touch events through tactile devices.
- **Timer**. To perform sets of Actions after a certain number of seconds.

These elements can provide basic coding knowledge without giving up complex game development, by only considering that the Game Loop implements the evaluation of the behaviour of every Actor on each iteration. Furthermore, in an attempt to enhance the simplification of the game development, the usage of logic expressions and complex data structures such as matrices, arrays or other complex structures like trees or graphs to create the Actor's Rules have been discarded.

2.3.3 The Game Editor

In order to democratize game development, it is necessary to aim at the easiest and most accessible way to present the development tools for game developers. For this reason, a game editor has been created, taking as its starting point some of the main features of the user interface literature and their subsequent integration within the whole SGE environment.

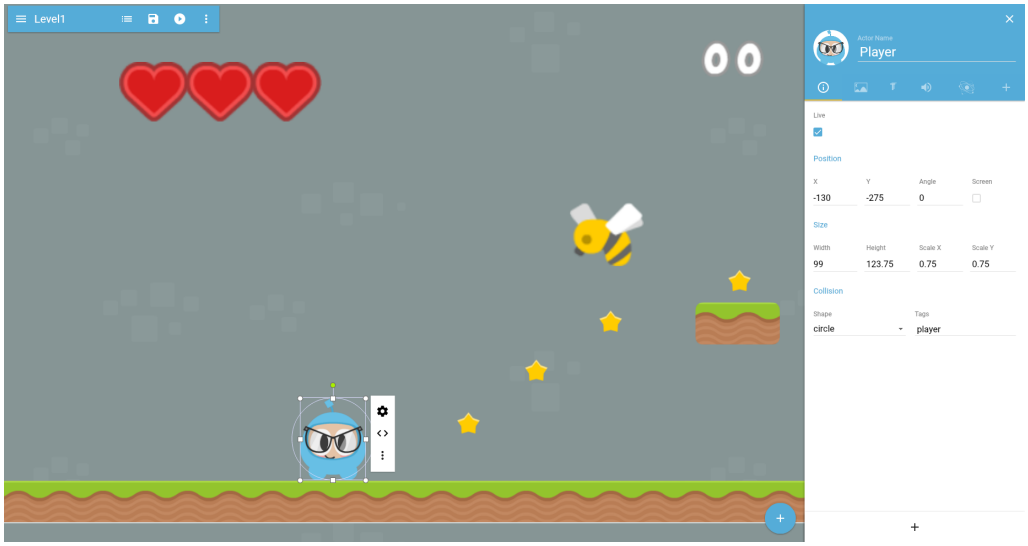

Figure 2.4: The Game Editor

The Scene Editor

The game editor has been developed by adopting the user interface design and interaction concepts behind a slide-show software [Tuf03]. This kind of application is conceived to be easy to use and has a very similar structure to games: several slides that can be compared to the game levels or Scenes, object placement and properties such as Actors and their Properties, and interactive event animations as the behaviour Rules.

Another design feature is related to its visual appearance. In this sense, Google Material Design [Goo19] has been the core of this interface specification, where the design definition is aimed at multi-device applications with fluid navigation. An example of this interface is shown in Figure 2.4 with a Platform Game. The Scene list for this game is available on the navigation menu. The canvas draws the environment designed for the first Scene called Level 1. The Actor, represented by the blue character, is selected and its gizmo and its associated graphical menu are visible. Through this menu, the user can access the Actor's properties and its Rules list. These features are also shown in Figure 2.4, where, after selecting the Actor's properties icon, a modular panel comes out from the right-hand side of the screen showing all the properties arranged on tabs in accordance with the grouping presented in Section 2.3.2.

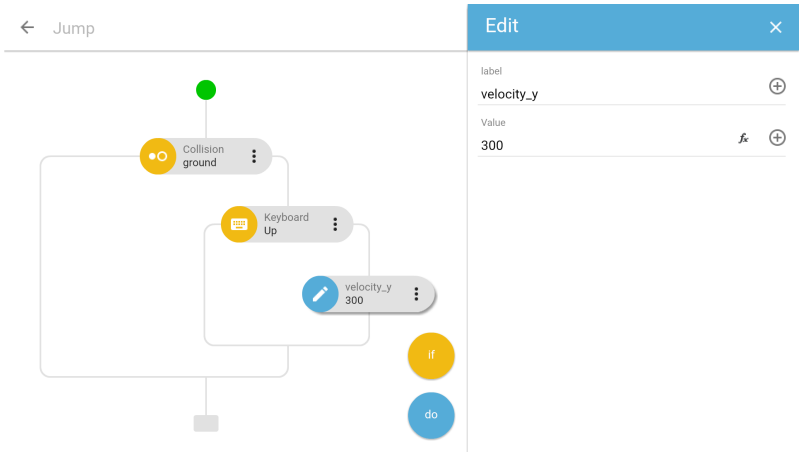

Figure 2.5: Appearance of the Rule editor when Edit is being configured to perform the *Jump* role. This example establishes the property *velocity_y* at 300 pixels/s

The Rule Editor

The roles performed by the Actors are defined in an edition tool environment, called Rule editor. This has been based on a simple decision tree, where the Actions and Conditions are arranged visually and configured until the desired behaviour is fulfilled.

Similarly to the Actor's Properties, this Rule editor can be accessed via the Actor's context menu. Figure 2.5 shows a decision tree editor, through which the behaviours can be composed by arranging the Actions and Conditions. These elements are accessible from two yellow and blue buttons, which serve as shortcuts to the set of Actions and Conditions, respectively. After selecting one of the elements, that panel comes out showing the Properties and the options determined for it. As an example of these features, Figure 2.5 shows a decision tree performing a Jump behaviour for a specific Actor. The Actor must be on the floor to be able to jump. Initially, a check is performed to determine whether the Actor is colliding with the floor and then, if the Up key is pressed, the jump action is produced by setting the Actor's y-axis velocity at 300 pixels per second. If any of these Conditions are not met, the flow will travel through its left branch and no Action will be performed. Each action or condition is configured by setting their Properties in a panel which appears on the right when it is selected. In order to facilitate understanding of the Rule for Jump, a pseudocode version is annexed in Algorithm 2.1.

Algorithm 2.1 Example of the associated pseudocode for the rule in Figure 2.5

```
If ( collision(Ground) )
    If ( KeyCode.Up )
        velocity_y = 300;
    End
End
```

2.4 Game Example: Candy Crush

Programming using SGE has some differences compared to the use of conventional programming paradigms. The creation of video games without the use of complex hierarchies and data structures such as vectors or matrices can be a challenge for expert programmers accustomed to using them in conventional programming languages. The impossibility of using loops when programming the rules within the actors, forces to assume the Game Loop as a way to perform the iterations. In this sense, programming video games with the proposed engine is more related to agent-oriented programming, where individual agents with similar properties interact with each other and with the environment to solve different types of problems. This way of programming allows to develop computational thinking in a more similar way to the interaction between people and can be easier for non-expert users. In order to illustrate the differences in the way a typical matrix problem can be solved, the creation of a Candy Crush-like game is proposed. It has been developed with SGE and Scratch, due to its orientation to the conventional learning of programming and its usage in some 2D game engines. This game is one of the 2D games with more complex mechanics that can be performed and its development with SGE demonstrates its great potential.

The Candy Crush game [CL16] is a match-three puzzle video game, in which the players have to swap candies on a game board to produce combinations of three or more with the same colour. This fact makes the candies disappear and allows the player to win points and to reach goals. Experimented computer developers usually use a matrix structure to store the game board, but a non-experienced user does not think in such complex data structures. Instead of using a matrix to handle the candies, each Candy becomes an Actor with its Properties and behaviour Rules that interacts with the rest of the Actors in the Scene. In this case, the Candies are organized in a matrix-like grid in the space, but there is no matrix as a data structure.

The size of the board for this implementation of the Candy Crush game is five

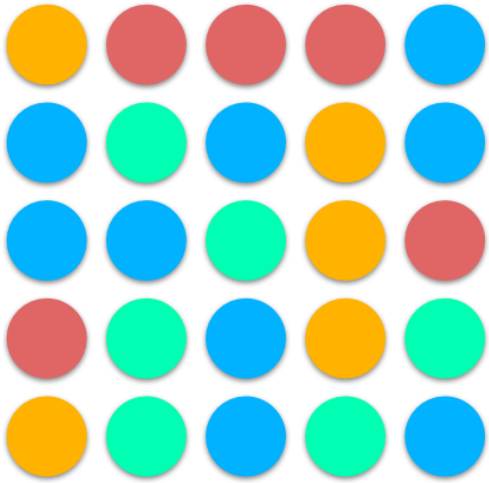

Figure 2.6: Example of Candy Crush initialization

elements in width and in height, and the Candies can display four different possible colours. Once the game board has been established with a random configuration, similar to the one shown in Figure 2.6, the Candy Crush game initialization is performed. The board is then filled with randomly coloured Candies.

Next, the implementation of the basic mechanics of the game using Scratch and later using SGE is considered.

2.4.1 Scratch Solution

The initialization procedure establishes the game board assigning a random texture with a specific colour to each Candy. The required code would have an instruction sequence similar to the one shown in Algorithm 2.2.

Algorithm 2.2 Candy's colour initialization in pseudocode for Scratch

```
i = 0;
j = 0;
Repeat Until ( i < rows )
    Repeat Until ( j < columns )
        color = random(1,4);
        Candy[i][j] = color;
        j = j + 1;
    End
    i = i + 1;
End
```

Continuing with the example, Algorithm 2.3 shows the function to remove three or more Candies in a row when they have the same colour. There, the algorithm checks each matrix row to see if there is a sequential combination of three or more Candies with the same colour. If that is the case, these Candies are removed.

Algorithm 2.3 Candy's checker and eraser in pseudocode for Scratch

```
i = 0;
j = 0;
Repeat Until ( i < rows )
    count = 1;
    remove = 0;
    Repeat Until ( j < columns )
        If ( Candy[i][j] == Candy[i][j + 1] )
            count = count + 1;
            If ( j == columns && count ≥ 3 )
                remove = count;
                position = j + 1;
            End
        Else
            If ( count ≥ 3 )
                remove = count;
                position = j;
            End
            count = 1
            Repeat ( remove > 0 )
                remove = remove - 1;
                delete( Candy[i][pos - remove] );
            End
        End
        j = j + 1
    End
    i = i + 1
End
```

2.4.2 SGE Solution

The same algorithm idea but performed with the SGE makes a significant change. Every Candy is represented by an Actor. Firstly, the developer has to set up the initial behaviour of Candy to create a random colour, essentially by making it choose between four images with four different colour textures. Then, it is only necessary

to copy and paste the Candy Actor as many times as needed until the grid shape shown in Figure 2.6 appears. This sets an arrangement of Actors who, during their initialization, choose a random image at the beginning of the Game. As can be appreciated, a matrix data structure is no longer necessary to initialize each Actor in the Game's creation process. Instead, each Candy initializes itself as it is presented in Algorithm 2.4.

Algorithm 2.4 Colour initialization of Candy

```
If ( initialization )
    image = image[random(0, 3)];
    initialization = false; // Initially the actor has this custom property at true
End
```

The algorithm to remove three or more consecutive Candies with the same colour is set up with three different Actors, which are not visible in the execution because they require no image.

- **Tracker**. This Actor is responsible for travelling from left to right crossing over every Candy in each row, checking its colour properties and counting the ones arranged consecutively. Initially, its colour value is empty and it obtains the one from the first Candy to collide with it. Its logic is presented in Algorithm 2.5.

Algorithm 2.5 Actor Tracker behaviour rule.

```
move ( x + 1 )
If ( colour == Candy.colour )
    counter++;
Else
    counter = 0;
    colour = Candy.colour;
End
If ( x > columns )
    destroy ( Tracker );
End
```

- **Eraser**. This Actor is spawned by Candy actors. If a Candy has a different colour than the one being tested and also the Tracker counter is 3 or greater, an Eraser is created. It receives information about how many Candies have to be destroyed. Its logic is presented in Algorithm 2.6.

Algorithm 2.6 Eraser Actor behaviour rule.

```
move ( x - 1 )
If ( collision(Candy) )
    destroy = destroy - 1;
    If ( destroy == 0 )
        destroy( Eraser );
    End
End
```

- **Candy**. In addition to that of the initialization, this Actor has two extra Rules. The first one is the colour checker: if the Tracker collides with the Candy, it checks whether it has the same colour as the one being counted, as presented in Algorithm 2.7. If this is the case, the Tracker counter is increased by one. If not, a check is performed to see if the Tracker counter is equal to or greater than 3, and if it is true an Eraser is created. The second Rule is in charge of destroying the Candy if an Eraser collides with it, as also shown in Algorithm 2.8.

Algorithm 2.7 Candy Actor behaviour Rule for the colour check.

```
If ( collision(Tracker) )
    If ( color == Tracker.color )
        Tracker.counter += 1;
    Else
        If ( Tracker.counter ≥ 3 )
            Eraser.destroy = Tracker.counter;
            spawn(Eraser);
        End
        Tracker.counter = 0;
        Tracker.colour = colour;
    End
End
```

As can be observed, in this way, complex data structures and their usage are avoided, breaking each role or Action in the Actor's behaviours down and making them interact with each other. As can be seen, both ways of reaching a solution are two comparable approaches to solving an algorithm problem, but conceptually it is a closer model to the target user's way of thinking [Win06, Win08].

Algorithm 2.8 Candy Actor behaviour Rule for the elimination of the Candy.

```
If ( collision(Eraser) )
    destroy(Candy);
End
```

If the algorithms are analyzed in each case, using Scratch and SGE, it can be seen that the behaviours that appear in each of the actors in SGE exist the Scratch (see Algorithm 2.3), however, in SGE, they appear separately and therefore so much easier to understand.

2.5 User Experience

To validate the game engine, an evaluation was carried out with children. The reason for using children as evaluators is that they are the perfect target user for these tools, that is, people familiar with technology but with the lack of the necessary knowledge to develop their own ideas in a videogame [HORB08]. The procedure consisted in having them develop their own videogame and asking them some broader questions to generate a deeper and more specific understanding of the problems encountered during the creation of a videogame [Gil92]. The children were organized by age into independent groups, each with a different arcade game and the evaluation was based on individual acceptance tests [Dav89, DV04]. In addition, a comparative test between SGE and Scratch was also performed. It was validated by the Wilcoxon Signed-Rank (WSR) test [LFH17].

2.5.1 Objectives and Hypothesis

The main assumption of this work is based on the perception that the game development process has the potential to be made easier, on the estimation that an arcade game can be made with a reduced set of Actions and Conditions, and also on the notion that the system's usage has to match the profiles of non-technical users. Implicitly, it is assumed that if these statements are met, then the SGE system has to be perceived as easy to use. It is necessary to verify that a tool such as the one presented here facilitates game creation for the users and it is found to be useful, along with the hypothesis that the rule editor is easy to understand. A summary of the objectives and hypothesis is presented in Table 2.2.

Table 2.2: Objectives and hypotheses for the experiment

Objectives	Hypotheses
O1 - The tool has to make game development easier	H1 - This tool makes the creation of games easier for non-programmers H2 - The tool is found to be useful
O2 - The tool has to be able to create arcade games using a reduced set of Actions and Conditions	H3 - The system of rules is easy to use H4 - The tool is easy to understand H5 - The tool is easier to use than other similar tools

2.5.2 Protocol

A total of one hundred and twenty children attending a summer camp related to technology served as the subjects for this experiment. Their ages ranged between seven and fourteen years. As regards gender, seventy-four of them were boys and forty-six were girls. Previously, their parents were informed about the aim and the method of the experiment, and they gave their informed consent for their sons and daughters to take part in this study. Regarding the level of previous experience of programming, all the children stated that they had never created a computer game.

The children had exactly twenty hours to work on their games. They were asked to design one of nine different arcade games according to their age. Figure 2.7 shows nine captures taken on the final versions of the games, the images are organized in the same order as described below, starting from the upper-left corner:

- **Asteroids**. A classic space and third-person shooter where the asteroids are subdivided after being hit.
- **Arkanoid**. A classic puzzle game where the ball has to bounce until there are no bricks left.
- **Cowboys vs Aliens**. Tower defence game where the Cowboys defend their land from the Aliens.
- **Car Racing**. A vertical scroll game where the goal is to advance as far as possible avoiding collisions with other cars and the road boundaries.
- **Tappy Plane**. A horizontal scroll game, the goal of which is to travel for as long as possible avoiding collisions with the environment.

Figure 2.7: Example of arcade games performed during the tests (sorted by rows according to their difficulty level)

- **Ducks**. A first-person shooter game based on the classic ones found in fairgrounds.
- **Abstract Adventure**. A classic platform game where the goal is to elude enemies and traps at the same time as every coin in the scene is gathered.
- **Blocks**. A physics game where the goal is to keep the player on the platform.
- **Combat**. A fighting game between two players.

The video games were developed individually. The children were arranged in groups of between ten and fifteen members each group working on one of the nine games that had previously been assigned to them. At the end of the experiment, they were asked to answer some simple questions about comfort with the tool and the understanding of the method. The response to each question was evaluated on a 5-point Likert scale [LFH17]. The questions were adapted to their age and were focused on the complexity of the game concept, in order to obtain a measurement of the Perceived Usefulness (PU) and Perceived Ease-of-Use (PEOU) ratio. Questions asked if they believed that using a particular system would require less effort and would enhance their job performance. The questions are presented in Table 2.3.

Table 2.3: Items and results for the PU and PEOU surveys

Questions		Average	S.D.
PU			
Q1	Has it been easy to create games?	3.66	0.77
Q2	Have you found it useful?	3.95	0.86
Q3	Have you understood the program's workflow?	4.00	0.77
Q4	Have you felt comfortable using the program?	4.44	0.56
Q5	Has it been easy to learn how to use the program?	3.98	0.61
PEOU			
Q6	Do you feel capable of making new games with the program?	4.31	0.80
Q7	Are you happy with the games you have created?	4.69	0.46
Q8	Would you like to continue making games like these ones?	4.45	0.74

Moreover, twenty-three of these children were asked to develop another video game using Scratch instead of SGE. At the end of the twenty hours' work, they filled in a questionnaire to compare SGE and Scratch. They answered a survey with three options: 1 represents a preference for SGE, −1 indicates that they prefer Scratch, and finally 0 indicates that they did not answer the question. The questions are shown in Table 2.4. These tests were assessed with the average and rank based on the WSR test.

2.6 Results

First, the evaluation of the SGE is presented in Table 2.3 and it is supported by the survey average and the standard deviation values. In addition, the distribution of the data can be observed graphically in Figure 2.8. Concerning the PU analysis, it can

Table 2.4: Comparative test on user acceptance between SGE and Scratch

Questions		Average	Signed Rank
S1	Which one has been the fastest to learn?	0.46	1.00
S2	Which one do you think you could create a new game with?	0.54	2.00
S3	Which one have you found easier to understand?	0.71	5.00
S4	Which one makes you feel happiest with the results?	0.83	6.00
S5	Which one do you think has been most useful to you?	0.58	4.00
S6	Which one has given you more facilities to perform a loop?	-0.54	-2.00
S7	Which would you like to continue working with?	0.88	7.00

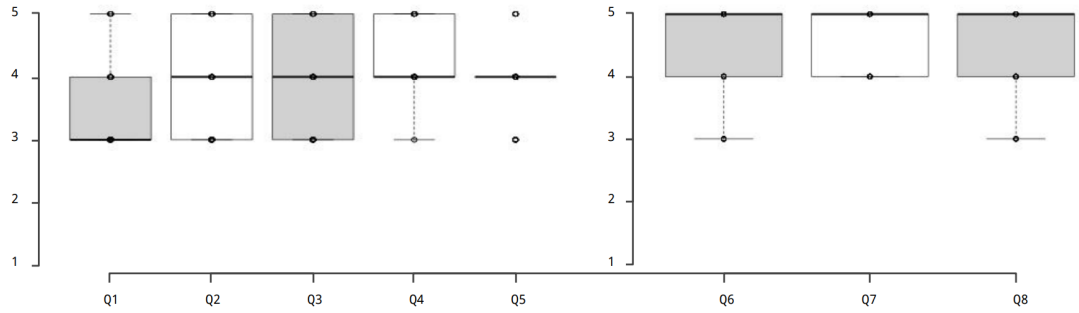

Figure 2.8: Graphical distribution of the data obtained for PU (left) and PEOU (right)

be seen that all the values vary in the range between 3 and 5, which represents a satisfactory evaluation for this game engine. Question Q1, concerning ease of use, is the one that has the worst average value: 3.66. This is because it is the first time the user tackles the design of a video game and considers that it is not such an easy task. However, they generally found the SGE system to be useful to perform this task (Q2) and quite easy to learn (Q5). The most remarkable data was obtained regarding question Q4, which indicates that they felt comfortable using this game engine. They also rated the program workflow of the SGE as easy to understand (Q3).

The results regarding PEOU were even better, confirming that the system's environment encourages the child's creativity for new game designs and their satisfaction with the games produced. The means of the data obtained in questions related to their attitude towards use or intention of use are in a range between 4 and 5. This fact confirms the proposed hypothesis because most of the children felt that the experience was worthwhile, rating each question in this category with a mean value of 4.48.

Concerning the evaluation of the tests that compare SGE and Scratch, all the data obtained are shown in Table 2.4. The average of this data is also shown. Regarding the WSR test, the signed-ranks of the data are calculated and shown in the rightmost column. These values have been analysed to evaluate the WSR test. Taking into account that twenty-three children ($n = 23$) answered these questions and using its proper significance level ($\alpha = 0.05$), the result of the test confirms that the data are statistically significant and so there is a difference between the two programming tools.

After analysing the results obtained, it can be said that the users feel the SGE

be a more useful and comfortable tool, thus highlighting the result of question S4 about the satisfaction with the game that was finally designed. Question S7 also confirms that they prefer to work with the SGE system than to use Scratch. However, the results obtained in question S6 show that Scratch has been perceived as an easier tool for creating loops in the video game. It was said that this concept is not explicitly included in the proposed game engine although it can be programmed by configuring the flowcharts. The fact that the users can develop their own loops in the Scratch code made the code that they built more understandable to them.

2.7 Discussion

The hypotheses set out regarding the proposed system have all been confirmed with the data obtained from the user tests. After the twenty hours of the experiment, each child was able to go home with his/her own video game completely finished. This fact confirms hypothesis H1 because the users were children without any knowledge of programming and they were capable of developing a complete 2D arcade video game. The fact that every game was completed on time along with the test results makes it possible to claim that the tool is easy to learn and to comprehend, which at the very least enforces hypotheses H3 and H4. Furthermore, the results also show that they generally agreed that the experience was perceived as helpful to them, which lends support to the validation of hypothesis H2. On another note, no significant usability problems were detected, although some syntax and vocabulary comprehension issues have been recognized as obstacles hindering the comprehension of the system.

Despite this, the children said that it was easy to use and understand. Most of them even stated that they were thrilled with their creations, and they were looking forward to starting their own ideas.

Furthermore, the results from the comparative study performed with Scratch and SGE proved that the SGE system is widely perceived as easier to use and to understand than other visual behavioural specification environments such as Scratch. These results reinforce the validation of H1 and support the validation of H5.

2.8 Conclusions and Future Work

This work presents a game engine, SGE, which addresses the aim of making the video game development process easier for people with non-technical profiles. Essentially, the way these interactive applications are developed has been changed from the traditional method of programming. Instead of using the data structures of some programming language, the SGE proposes a reduced set of Actions and Conditions to develop arcade games. These elements are combined in flow charts that determine the actions carried out by the actors, the main component of this game development system.

The game specification has been restricted in order to achieve a reduced game engine architecture and a straightforward game editor. The game development has been performed without relying on matrices, loops and other complex data structures and through a reduced set of Actions and Conditions. Some of the tests performed have been conducted in order to evaluate the simplicity of SGE in performing this task. The results demonstrate that, by using this game engine, users are able to develop games in an easier way. The game engine proposed in this research has been tested by children without any knowledge of programming, who develop games using this system and with the popular visual programming tool Scratch. Results obtained from these experiments show that SGE is perceived as a useful and easy-to-use tool for game development and for learning to programme, even when compared with Scratch.

The experience acquired during this work will push us in the future to continue the development of this project on a 3D game engine. Similarly, it will have to be easy to use and ready to enable non-programming people to develop games with visual behaviour descriptors. Currently, it is already easy to create 3D environments or very simple games with editors such as Minecraft but it is still necessary to have advanced knowledge to be able to create games with development frameworks such as Unity. In this sense, bridging the gap between these two types of editors is going to be one of the main research topics from now on, in order to allow users to build 3D arcade games without any knowledge of coding.

Part III

Game Engine Formalization Using Multi-agent Systems

Chapter 3

A Game Engine to Make Games as Multi-agent Systems

Publication

Marín-Lora, Carlos., Chover, Miguel., Sotoca, Jose M., & García, Luis. A. (2020). A Game Engine to Make Games as Multi-agent Systems. Advances in Engineering Software, 140, 102732. (Q1).
https://doi.org/10.1016/j.advengsoft.2019.102732.

Abstract

Video games are applications that present design patterns that resemble multi-agent systems. Game objects or actors are like autonomous agents that interact with each other to describe complex systems. The purpose of this work is to develop a game engine to build games as multi-agent systems. The actors or game engine agents have a set of properties and behaviour rules with the end to interact with the environment of the game. The behaviour definition is established through a formal semantic based on predicate logic. The proposed engine tries to fulfil the basic requirements of the multi-agent systems, by adjusting the characteristics of the system, without affecting its potential. Finally, a set of games are introduced to validate the operation and possibilities of the engine.

Keywords

Game development, Game engine, Multi-agent Systems

3.1 Introduction

Game engines are tools to facilitate video game development. They were conceived to generalise and reuse properties, methods and procedures common to the majority of games [Gre18]. Through them, designers can generate different game mechanics using the same components and scripts. Currently, the vast majority of commercial game engines are designed with equivalent organisational paradigms and similar software specification patterns [Nys14], increasing their capabilities and performance. Nevertheless, different works show the need to establish a standard specification of the game engine architectures [AEMC08, AS10].

When analysing the games produced by these engines, it is perceived that the design structures resemble the modelling patterns of multi-agent systems (MAS) [DKJ18]. The study of these game engines and their characteristics suggest that there is a close relationship between the concepts of game and MAS. This is appreciated in the game elements of the agents' definition [SCT03], their relations [Bis08] and its communication protocols and cooperation mechanisms [GSF13]. In fact, there are several related topics, paradigms or applications where the MAS are used for the control of automatic processes and dynamic systems [OS06], or mechanisms of cooperation and consensus [OSFM07].

With all this, a game engine can be formally defined from the knowledge of a MAS. The initial hypothesis is that a game can be defined and specified as a system of agents interacting with each other. In this regard, the proposed game engine must be able to generate functional games that satisfy the properties of MAS. For this purpose, this work proposes a game engine for the creation of games defined as MAS. The engine's formalisation follows the mathematical notation stated by M. Wooldridge [Woo09] in order to define the structure of the game engine and to fulfil with the requirements of the proposed system.

The manuscript is organised as follows. Section 4.2 presents the current state of the art on game engines and their relation with MAS. In Section 4.3, the formal definition of a MAS is introduced. Later in Section 4.4, the proposed game engine is defined by following the formal specification shown in the previous section, along

with the behaviour definition system for the game engine agents. In Section 4.5, the concepts stated in the previous section and its implications are discussed. In Section 4.6, a series of use cases are presented to demonstrate the potential of the tool. Finally, in Section 4.7, several conclusions and the possible improvements of the proposed game engine are presented.

3.2 Game Engines and Multi-agent System

The term game engine appeared in the mid-1990s in reference to the architecture of independent software components that defined video games such as Doom from IdSoftware [Gre18]. This architecture gained value as game developers began to create generic modules that facilitate the creation of new games, and thus reduce development times. In a game engine, the modules are the subsystems responsible for executing specific tasks, such as the drawing, the user interactions or the game physics. However, some of these modules are not easy to standardise, either because they involve complex systems that can be considered as complete engines by themselves, or because of their embedded relationships with the mechanics of each game [Doh03, LJ02]. The module responsible for managing the game mechanics is the logic module and the decision-making system. Its function is implicitly related to the character's internal processes associated with their autonomous potential actions [Mil19]. The range of available actions is dependent on game mechanics and must be established by the game designer. A game object can have simple rules to determine what to do next, such as the case in which a Non-Playable Character (NPC) follows the player character as long as it is within a range of action.

These autonomous behaviours are directly dependent on the communications between the entities of the game, either sending information about the game properties or about the game objects properties. It is at this point where the relationship between games and MAS becomes evident. There are several cases in the literature where different perspectives address the relationship between game concept and MAS. Thus, some present the relationship between MASs and the game mechanics design, emphasising the industry's tendency to relate natural language and the game mechanics definition during the creation of games [DWvDH09], while others propose a framework for the creation of agents for serious games [JFP10]. Further works present a MAS to model architecture for Massive Multiplayer Online Game (MMOG) games [ACB08, ATE+12], where the real-time interactivity of multiple game

agents prevails. Other authors show a system that tries to integrate virtual worlds with a multi-agent platform through an interface [GSF13]. Also, the MASs have been used to expose a virtual environment where agents communicate autonomously with the player as a 3D chat [GQCC06, GQC+07a, GQCC10], and subsequently it was extended for the implementation of a virtual fair [RGR+15, RCRG06, RRPCC12]. In addition, they have been employed to raise a distribution where it is possible to build multi-player systems based on intelligent agents [SIB11], to propose a MAS to manage multi-user mechanics in a tournament game [AMS+01], and to present a system based on agents that control the parameters of the game according to the objectives to be achieved [PB13]. Finally, in terms of the relationship between MAS and game engines, a MAS based on the Unity game engine has been developed [Uni22], generating a three-dimensional search behaviour simulation of multiple agents in the context of a passenger airport [BARHN14], and also a system where agents learn autonomously to play multiple games without human intervention [FB08].

Nevertheless, the approaches described above add MAS features to specific games or specific genres, but no one of them introduces a functional approach of MAS as the core of the game development. In this sense, the MAS would fulfil one of the original purposes of the game engines: the flexibility in the procedures and the modules aggregation.

3.3 Multi-agent System Features

Following the agent definition proposed by M. Wooldridge [Woo09], an agent is a computer system that is situated in an environment, and it can perform autonomous actions in this environment in order to meet its design objectives. More specifically, in the general definition of MAS, it is necessary to take into account specific features when carrying out its formalisation:

- The environment to which the agents belong can be in any of the discrete states of a finite set of states E = $[e_0, e_1, e_2, \cdots]$.
- The agents have a set of possible actions available with the ability to transform their environment Ac = $[\alpha_0, \alpha_1, \alpha_2, \cdots]$.
- The run r of an agent on its environment is the interlayered sequence of actions and environment states $r : e_0 \rightarrow^{\alpha_0} e_1 \rightarrow^{\alpha_1} e_2 \rightarrow^{\alpha_2} \cdots e_{u-1} \rightarrow^{\alpha_{u-1}} e_u$.

- The set of all possible runs is R, where R^{Ac} represents the subset of R that ends with an action, and R^{E} represents the subset of R that ends with a state of the environment. The members of R are represented as R = $[r_0, r_1, \cdots]$.
- The state transformation function τ introduces the effect of the actions of an agent on an environment τ: $R^{Ac} \rightarrow \wp(E)$ [FHMV04].

Thus, the following definitions can be established:

D1. An Env environment is defined as a triplet Env = $\langle E, e_0, \tau \rangle$, where E is a set of states, $e_0 \in E$ is an initial state and τ is the state transformation function.

D2. An Ag agent is defined as a function Ag: $R^{E} \rightarrow Ac$ and establishes a correspondence between runs and actions. It is assumed that these runs end in an environment state [RS94].

In this way, an agent makes the decision on what action to take based on the history of the system it has witnessed to date. The set of all agents Ag in a system is represented as AG and the set of runs of an agent Ag over the environment Env is represented as $R(Ag, Env)$.

D3. A purely reactive agent is defined as a function Ag: $E \rightarrow Ac$ [GN12], which indicates that they make the decision based only on the present state of the environment.

D4. An agent with perception is considered as such when it is composed of perception functions and actions, as Ag = $\langle see, action \rangle$. Where see: $E \rightarrow Per$ maps environment states to percepts and $action$: $Per^* \rightarrow Ac$ maps sequences of percepts to actions.

D5. Agents with an internal state are those that have an internal data structure I used to store information, where I = $[i_0, i_1, i_2, \cdots]$ is the set of all internal states of an agent. In this case, the action function is defined as a correspondence $action$: $I \rightarrow Ac$. Additionally, there is a next function that generates new internal states from perceptions $next$: $I \times Per \rightarrow I$. The action to be performed by the agent will, therefore, be $action(next(i, see(e)))$. After completing the action, the agent re-enters the perception cycle of the environment, it updates its status through next, and it selects the action to be taken with action.

Agents are goal-oriented, as they perform actions to satisfy some goal. The way these goals are represented is by using task predicates Ψ.

D6. Let $\Psi(r)$ indicates that the run $r \in R$ satisfies the predicate Ψ as $R_\Psi(Ag, Env)$ = $\{r|r \in R(Ag, Env)$ and $\Psi(r)\}$. Then, an agent Ag succeeds in the task $\langle Env, \Psi\rangle$ if $R_\Psi(Ag, Env)$ = $R(Ag, Env)$. In other words, Ag succeeds in $\langle Env, \Psi\rangle$ if every run of Ag in Env satisfies Ψ. Thereby, an agent just runs its right tasks.

3.4 The Game Engine

To define the structure of the proposed game engine, this work focuses on the definition of the elements that make up the system and the behaviour specification model generated for these elements. In this sense, formal correspondences are established between the games and MAS, to demonstrate that the engine allows specifying games defined as a MAS. The presented game engine makes possible the creation of 2D video games.

3.4.1 The Game

The engine allows defining the environment of the game where the action is performed. The run of the game $R(AG, Env)$, represents the run of all agents AG on an environment Env. This environment Env is responsible for storing the game states E through the properties of it. The initial state e_0 determines the set of initial properties. The properties defining the game state are:

- **Camera**. It includes the camera position, rotation and zoom on the environment where the action takes place.
- **Audio**. A set of parameters to define and modulate the game's sounds such as pan, volume, start time and loop.
- **Physics**. They define, for instance, the intensity of the gravity force on the game, its influence on the actors and their physical properties. The game engine, therefore, is able to build games with realistic physics.

The agents AG of the system are described below, and they are called actors. These actors can transform the environment states according to their tasks and through the transformation operator .

3.4.2 The Actor

The actors are responsible for running the actions Ac on the environment Env. In the proposed engine, the actors are purely reactive agents [GN12], since they make the decision-making process with consideration of the present. Also, the actors are agents with perception [Woo09, RS94], since in every moment they are watching the game state to evaluate it and to act accordingly. The actors have internal state I = $[i_0, i_1, i_2, \cdots]$, which is initially defined with a set of properties. The basic properties of the actors are classified into the following categories:

- **Geometry**. It includes properties related to the position, rotation and scale of the actors.
- **Render**. In this case, it includes the actor image and its related properties: opacity, flip, scroll, tile and tinting colour.
- **Text**. The actors can write text on the screen, according to a font, size, colour and style parameters. They can also write the value of any property from the game or from any actor.
- **Audio**. The specific sound the actor plays and the set up of its properties: start time, volume, pan and loop.
- **Physics**. The physical features for the actors such as speed, angular velocity and material properties: density, friction and restitution.
- **New**. Also, the actors can incorporate new knowledge as new properties that expand their internal state.

In the game engine, the perception of the environment is defined by the evaluation of the actor's physical behaviour and by the signals derived from the interaction with the user. Based on the actor's state and the perception of the environment, it is produced an evaluation of the actor's knowledge, generating a change in the game state or in the actors' state that allows selecting the actions α associated with its tasks. Logical predicates Ψ define the specification of the tasks performed by the actors. The actors run their tasks in the system if $\exists r \in R(Ag, Env)$ such that the predicate $\Psi(r)$ is satisfied. Figure 3.1 shows the interaction cycle between the actors and the game.

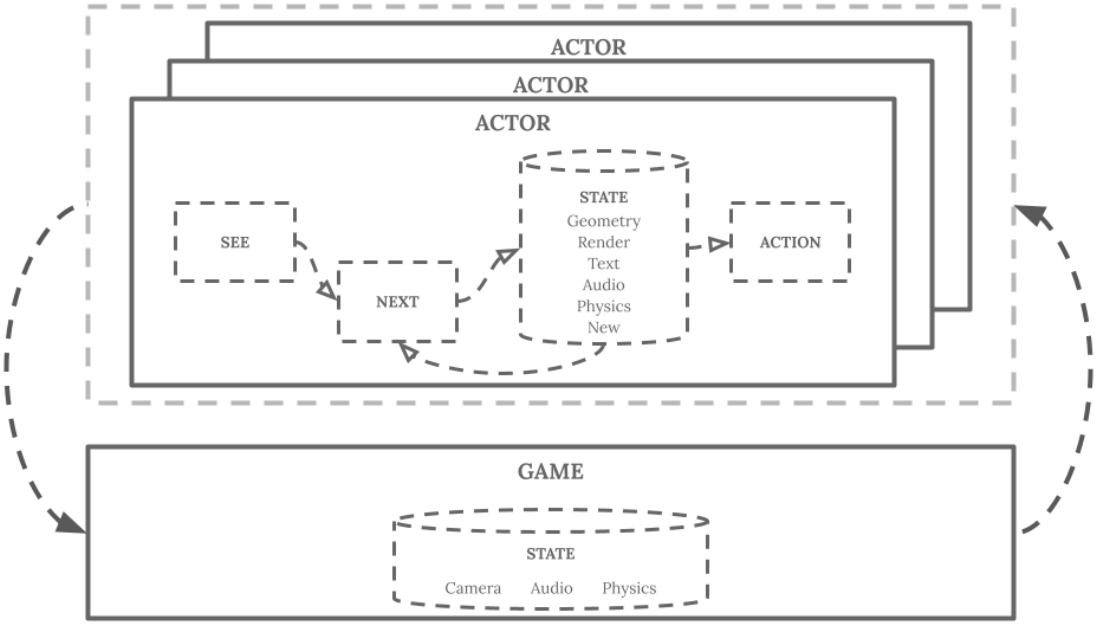

Figure 3.1: The actors and their interaction cycle with the game.

3.4.3 Behaviour Specification

The behaviour specification system is responsible for telling the actors what tasks to perform, based on the game state and the internal state of the actors, following a predefined semantic. This semantic is defined by a syntax of logical and non-logical symbols. Logical symbols include logical connectives and variables while non-logical symbols include predicates and functions [BLR92]. The domain for this interpretation is the set of all the actors and the game.

The predicates Ψ defined in first-order logic specify the actor's behaviour rules. In this way, the tasks that an actor might perform are organised into predicate formulas

$$\{\Psi_0, \Psi_1, \Psi_2, \cdots\}$$

where each formula can contain the following predicate structures:

- **Condition structure**: It is modelled by a predicate sequence model based on the structure of the IF-THEN-ELSE rules [Kar88].

$$(If \rightarrow \Psi) \wedge (\neg If \rightarrow \theta)$$

where, If is a conditional literal predicate and both Ψ and θ are sequences of new predicates to be evaluated if the condition is met or if it is not met, respectively.

- **Action structure**: It is composed of an atomic element that includes a single literal predicate Do.

Conditional Predicate If

The conditional predicate represents the evaluation element of a condition in the decision-making process. In this game engine, the evaluation of the condition is based on a function that assesses the relationship between system entities.

$$If(function(parameters))$$

The parameters can contain variables as arithmetic expressions, which can include game or actor properties, mathematical functions and constant numerical values.

This function can be of the following types:

- **Compare**: This function compares numerical and boolean values. The relationships are established by a set of comparison operators whose elements can be greater, greater or equal, equal, less or equal and less.

$$compare(x, y, label)$$
$$x, y \in \mathbb{R}$$
$$label \in [greater, greaterEqual, equal, lessEqual, less]$$

- **Timer**: This function handles the system's timing. It determines if a specific time x has expired so that the evaluation of the function is true.

$$timer(x)$$
$$x \in \mathbb{R}$$

- **Pointer**: This function controls the pointer-events on screen, where x represents the coordinates for those events on the game space.

$$pointer(x)$$
$$x \in \mathbb{R}$$

- **Keyboard**: This function oversees the system's keyboard events, where x is the code for a specific key and label represents the event mode.

$$keyboard(x, label)$$
$$x \in \mathbb{R}$$
$$label \in [press, release]$$

- **Collide**: This function watches the collision detection between game actors, where Ag is the actor to report a collision.

$$collide(Ag)$$
$$Ag \in AG$$

Action Predicate Do

An action is defined as a specific behaviour to be performed by an actor. In the game engine, the actions are the non-logical function elements that represent the actions $\alpha \in Ac$ of the system. The actions can contain variables as arithmetic expressions in the same way as for the conditions.

This set of actions is based on the *Create*, *Read*, *Update* and *Delete* (CRUD) functions of information persistence on databases [Dai10] applied to the actors. From these actions, the system can generate more complex actions that increase the abstraction level of the behaviour specification. With all this, the set of actions Ac available in the system consists of the following actions α:

- **Create**: It creates a new actor in the game as a copy of an existing actor Ag in the environment.

$$create(Ag)$$
$$Ag \in AG$$

- **Read**: It allows reading the information of a property that may belong to an actor $Ag \in AG$ or to the environment Env of the game.

$$read(property)$$

- **Update**: It modifies the value of a property that can belong to an actor Ag or the environment Env of the game. The new value is determined from the evaluation of an arithmetic expression.

$$update(property, expression)$$

- **Delete**: It removes an actor Ag from the game environment.

$$delete(Ag)$$
$$Ag \in AG$$

3.5 Discussion

The proposed engine allows creating a wide variety of video games but also has some features that are interesting to analyse. In the first place, it has been tried to design a system that formally creates video games, based on the MAS analysis. Thereby, the engine has the essential features to create a large variety of games without other attributes from the conventional 2D game engines which, in experimentation have been proved as not necessary.

Next, it is necessary to emphasise that the games are made with a single type of actor and that there are no hierarchical relations between them. All the elements that define a game: the markers, the player, the NPC and so on, have the same structure of properties and behaviour rules. Also, the use of a scene graph is not necessary, which simplifies the engine internal architecture and the design of the game.

The properties defined for the game environment and the actors allow only data types such as text strings, numbers and booleans. There are no complex data structures such as vectors or matrices, which are not necessary for the creation of most arcade games. For instance, a CandyCrush-like game seems to require a matrix for its setup, but it can be created from a MAS perspective by using multiple agents arranged in rows and columns, including their suited properties and behaviour rules.

Besides that, if the behaviour specification language for the actors is analysed in detail, it can be verified that it is possible to define game behaviour using a set of predicate formulas of just five conditions and four actions, as it has been stated in the previous section. This is one of the main features of the proposed engine since it allows the creation of games without using complex scripting languages as in other game engines. Also, an analysis of this predicate language brings forward that logical operators are not used and it is only necessary to use the IF-THEN-ELSE structure in a nested way to define behaviour. Not even loops are required, since the

game cycle itself performs a sequential evaluation of each of the actor's behaviour rules per cycle and it can, therefore, be avoided.

In the following section, it is presented a detailed description of some video games developed with the engine.

3.6 Use Cases

In order to test if the proposed game engine is capable of generating fully functional games that comply with the properties of a MAS, the game logic for three games has been constructed using the formulation presented in the previous sections. Specifically, the first of these games is the Wolf-Sheep Predation, which is based on a classic MAS problem; the other two are classic arcade games: Frogger and Pac-Man. In these games, the origin of the coordinate system is in the centre of the screen and it has positive and negative values.

Finally, an additional set of different arcade games have been created with the engine to demonstrate its potential.

3.6.1 Wolf-Sheep Predation

In this first game, the mechanics consist of a set of actors representing Wolves and Sheeps trying to survive in the game environment. These actors move arbitrarily through the stage, and every movement implies an expenditure of energy. Also, there is another actor representing the Grass, spread evenly on the stage. When a collision occurs between a Sheep actor and a Grass actor, the latter is destroyed, and the energy of the Sheep actor increases. Similarly, when a Wolf actor collides with a Sheep actor, it is also destroyed and the Wolf actor's energy increases. The reproduction of both actors occurs from time to time, depending on the game settings, as long as they have a sufficient level of energy.

Additionally, in the implementation of this game, there are three auxiliary actors in charge of generating the initial distribution of the three main agents. These auxiliary actors spawn new Sheep, Wolf and Grass actors initialised with an energy value and an initial arbitrary movement. Due to these three actors are not involved in the central dynamics of the game, they are not explained in the following game's algorithms.

Sheep

At first, the Sheep actors energy value is set at 300, and also their ρ_1 rule gives them an initial arbitrary movement.

Algorithm 3.1 Sheep Initialization

$initialization(\rho_1)$	`If(`$compare(init, true, equal)$`)` $\rightarrow$
	`Do(`$update(velocity, random(-100, 100))$`)` $\wedge$
	`Do(`$update(init, false)$`)`

Besides that, when they collide with the limits of the screen, they change their direction, as it is indicated by their ρ_2 and ρ_3 rules. When colliding with a Wolf actor, the specific Sheep actor must be destroyed according to its ρ_4 rule.

Algorithm 3.2 Sheep Collisions

$wallXCollision(\rho_2)$	`If(`$collide(wallX)$`)` $\rightarrow$
	`Do(`$update(velocity.x, 1 \times velocity.x)$`)`
$wallYCollision(\rho_3)$	`If(`$collide(wallY)$`)` $\rightarrow$
	`Do(`$update(velocity.x, 1 \times velocity.y)$`)`
$wolfCollision(\rho_4)$	`If(`$collide(wolf)$`)` $\rightarrow$
	`Do(`$delete(sheep)$`)`

In respect of the Sheep actor reproduction, the ρ_5 rule controls the spawn of the actor and the energy loss. It is dependent on a random expression parametrized to trigger it on the 2% of the cases. If it happens, the Sheep actor would lose 50 energy units. In order to recover energy, it has to "eat" Grass actors to gain 50 energy units.

Algorithm 3.3 Sheep Life Cycle

$reproduce(\rho_5)$	`If(`$compare(random(0, 100), 2, less)$`)` $\rightarrow$
	`If(`$compare(energy, 50, greaterEqual)$`)` $\rightarrow$
	`Do(`$create(sheep)$`)` $\wedge$
	`Do(`$update(energy, energy - 50)$`)`
$grassCollision(\rho_6)$	`If(`$collide(grass)$`)` $\rightarrow$
	`Do(`$update(energy, energy + 50)$`)`

This feeding process is essential, since each displacement implies an energy

expenditure, specifically of an energy unit, as it can be seen in its ρ_7 rule. By extension, if the energy reaches zero, the Sheep actor dies according to its ρ_8 rule.

Algorithm 3.4 Sheep Stamina

$walk(\rho_7)$	$\texttt{Do}(update(energy, energy - 1)) \wedge$
$death(\rho_8)$	$\texttt{If}(compare(energy, 0, lessEqual)) \rightarrow$
	$\quad \texttt{Do}(delete(sheep))$

Wolf

Similarly to the Sheep actors, the Wolf actors have an initial energy value equal to 300 and their initial movement is determined by their ρ_1 rule.

Algorithm 3.5 Wolf Initialization

$initialization(\rho_1)$	$\texttt{If}(compare(init, true, equal)) \rightarrow$
	$\quad \texttt{Do}(update(velocity, random(100, 100))) \wedge$
	$\quad \texttt{Do}(update(init, false))$

The rest of its rules are identical to the Sheep actor ones, but exchanging the Grass actors as feed for Sheep actors as it is indicated by their ρ_4 rule.

Algorithm 3.6 Wolf Rules

$wallXCollision(\rho_2)$	$\texttt{If}(collide(wallX)) \rightarrow$
	$\quad \texttt{Do}(update(velocity.x, 1 \times velocity.x))$
$wallYCollision(\rho_3)$	$\texttt{If}(collide(wallY)) \rightarrow$
	$\quad \texttt{Do}(update(velocity.x, 1 \times velocity.y))$
$sheepCollision(\rho_4)$	$\texttt{If}(collide(sheep)) \rightarrow$
	$\quad \texttt{Do}(update(energy, energy + 50))$
$reproduce(\rho_5)$	$\texttt{If}(compare(random(0, 100), 2, less)) \rightarrow$
	$\quad \texttt{If}(compare(energy, 50, greaterEqual)) \rightarrow$
	$\quad\quad \texttt{Do}(create(wolf)) \wedge$
	$\quad\quad \texttt{Do}(update(energy, energy - 50))$
$walk(\rho_6)$	$\texttt{Do}(update(energy, energy - 1))$
$death(\rho_7)$	$\texttt{If}(compare(energy, 0, lessEqual)) \rightarrow$
	$\quad \texttt{Do}(delete(wolf))$

Figure 3.2: A capture of the Wolf-Sheep Predation game.

Grass

In this case, the purpose of this actor is to feed the Sheep actors. Thus, through its ρ_1 rule, the Grass actors detect when a Sheep actor collides with them, and in that case, they proceed to destroy themselves.

Algorithm 3.7 Grass death

$sheepCollision(\rho_1)$	`If(`$collide(sheep)$`)` $\rightarrow$ `Do(`$delete(grass)$`)`

With the Wolf-Sheep Predation case, it has become clear that the proposed system is able to generate autonomous entities capable of generating complex behaviours from simple rules. It is important to highlight that the dynamics of the three principal actors of this game present similar behaviours, where the same event is treated separately by each involved agent, according to the role assigned to it in the predator-prey relationship. A screenshot of the game's run can be seen in Figure 3.2.

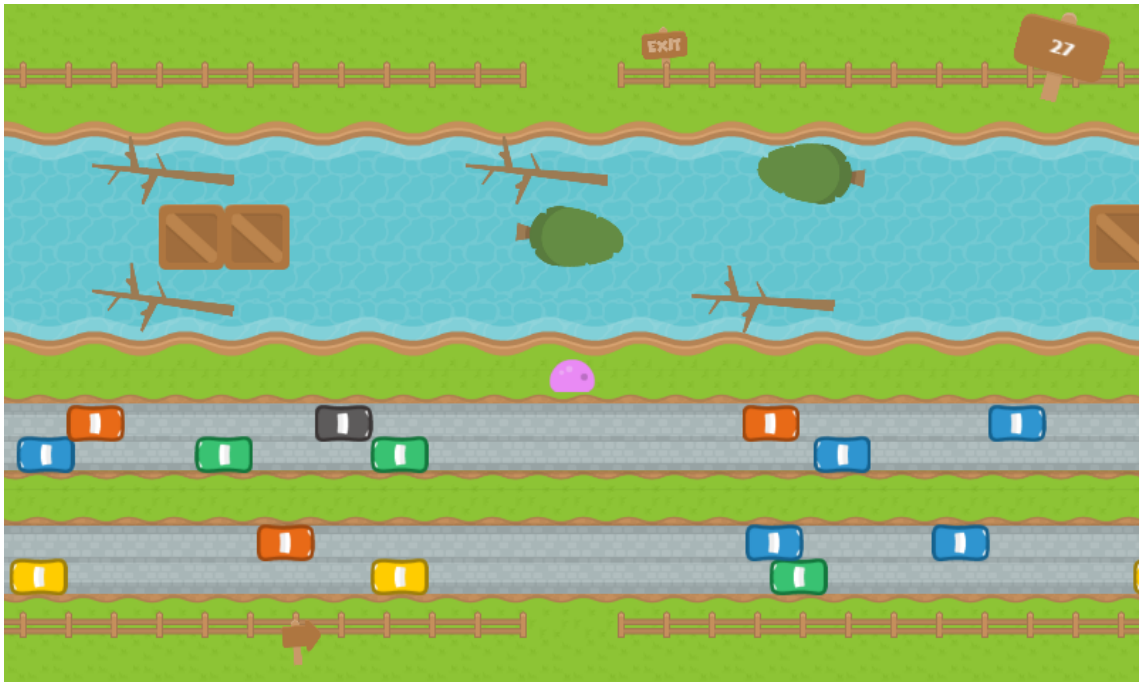

Figure 3.3: A capture of the Frogger game.

3.6.2 Frogger

The Frogger game consists of a Frog actor trying to cross a road with a series of Car actors passing by and a river with Trunk actors that must be used to reach the other side. The Car actors and the Trunk actors are initialised with a constant speed that makes them move from right to left. Also, there is an actor representing the Water area and two other auxiliary actors working as Car and Trunk generators. As it is shown in Figure 3.3, there are five lanes through which Car and Trunks actors circulate. Each one of these lines has an auxiliary actor that spawns Car or Trunk actors based on a timer function, which is controlled by a random expression to make its appearance in the game less predictable.

Frog

The Frog can move in any direction that the user determines with its ρ_1, ρ_2, ρ_3 and ρ_4 rules.

Algorithm 3.8 Frog movement

$leftKey(\rho_1)$	$\texttt{If}(keyboard(left, press)) \rightarrow$
	$\quad \texttt{Do}(update(position.x, position.x - 10)) \wedge$
	$\quad \texttt{Do}(update(angle, 270)))$
$rightKey(\rho_2)$	$\texttt{If}(keyboard(right, press)) \rightarrow$
	$\quad \texttt{Do}(update(position.x, position.x + 10)) \wedge$
	$\quad \texttt{Do}(update(angle, 90)))$
$upKey(\rho_3)$	$\texttt{If}(keyboard(up, press)) \rightarrow$
	$\quad \texttt{Do}(update(position.y, position.y + 10)) \wedge$
	$\quad \texttt{Do}(update(angle, 180)))$
$downKey(\rho_4)$	$\texttt{If}(keyboard(down, press)) \rightarrow$
	$\quad \texttt{Do}(update(position.y, position.y - 10)) \wedge$
	$\quad \texttt{Do}(update(angle, 0)))$

If it collides with a Car actor, it is removed from the game as it is set up in its ρ_5 rule.

Algorithm 3.9 Frog collision with Car

$destroyByCar(\rho_5)$	$\texttt{If}(collide(car)) \rightarrow$
	$\quad \texttt{Do}(delete(frog))$

In the same way, with its ρ_6 rule, if it collides with the Water actor and is not in contact with a Trunk actor, it is also deleted from the stage.

Algorithm 3.10 Frog collision with Water

$destroyByDrowning(\rho_6)$	$\texttt{If}(collide(water)) \rightarrow$
	$\quad \neg\, \texttt{If}(collide(trunk)) \rightarrow$
	$\quad\quad \texttt{Do}(delete(frog))$

When it lands on a Trunk actor, as it is indicated by its ρ_7 rule, it inherits the movement from right to left from the Trunk actor.

Algorithm 3.11 Frog collision with Trunk

$trunkCollision(\rho_7)$	$\texttt{If}(collide(trunk)) \rightarrow$
	$\quad \texttt{Do}(update(velocity.x, 100))$

Car

Cars always move from right to left. If they reach the left edge of the screen, they are eliminated from the game as indicated by their ρ_1 rule.

Algorithm 3.12 Car life cycle

$destroy(\rho_1)$	`If(`$compare(position.x, -screen.width/2 - width, lessEqual)$`) →` `Do(`$delete(car)$`)`

Trunk

As in the previous actor, it moves from right to left until it is entirely out of the screen as it can be seen at its ρ_1 rule.

Algorithm 3.13 Trunk life cycle

$destroy(\rho_1)$	`If(`$compare(position.x, -screen.width/2 - width, lessEqual)$`) →` `Do(`$delete(trunk)$`)`

With the Frogger game, a classic arcade game has been generated where all entities have completely autonomous behaviour and whose orchestration generates an elaborate game. Simple actor dynamics determine the mechanics of this game based on simple movements and interactions between the Frog actor and the other actors present in the game. In Figure 3.3, a capture of the Frogger game during its run on the game engine is shown.

3.6.3 Pac-Man

The Pac-Man actor is in a maze along with four Ghost actors. These actors can move in any direction as long as they do not collide with the maze walls. If the Pac-Man actor collides with a Ghost actor, it loses the game. Throughout the alleys of the maze, a set of Food actors are arranged to provide points to the Pac-Man actor after a collision with them. Also, there is a set of Coin actors to allow the Pac-Man actor to chase and eliminate the Ghost actors. When the Pac-Man actor

collides with a Coin actor, a short time begins in which it can chase the Ghost actors and destroy them.

Pac-Man

The Pac-Man actor is initialised with zero points and with a constant speed. The user controls the direction of its movement through its ρ_1, ρ_2, ρ_3 and ρ_4 rules.

Algorithm 3.14 Player movement

$leftKey(\rho_1)$	`If(`$keyboard(left, press)$`)` $\rightarrow$
	`Do(`$update(velocity.x, =100)$`)` $\wedge$
	`Do(`$update(velocity.y, 0)$`)` $\wedge$
	`Do(`$update(angle, 180)$`))`
$rightKey(\rho_2)$	`If(`$keyboard(right, press)$`)` $\rightarrow$
	`Do(`$update(velocity.x, 100)$ $\wedge$
	`Do(`$update(velocity.y, 0)$`)` $\wedge$
	`Do(`$update(angle, 0)$`))`
$upKey(\rho_3)$	`If(`$keyboard(up, press)$`)` $\rightarrow$
	`Do(`$update(velocity.x, 0)$`)` $\wedge$
	`Do(`$update(velocity.y, 100)$`)` $\wedge$
	`Do(`$update(angle, 90)$`))`
$downKey(\rho_4)$	`If(`$keyboard(down, press)$`)` $\rightarrow$
	`Do(`$update(velocity.x, 0)$`)` $\wedge$
	`Do(`$update(velocity.y, =100)$`)` $\wedge$
	`Do(`$update(angle, 270)$`))`

If it collides with the maze walls, it stops as indicated by its ρ_5 rule.

Algorithm 3.15 Wall limits

$wallCollision(\rho_5)$	`If(`$collide(wall)$`)` $\rightarrow$
	`Do(`$update(velocity, 0)$`))`

Besides that, if it collides with a Food actor, its points property increases as it is marked by its ρ_6 rule. However, if it collides with a Coin actor, as its ρ_7 rule indicates, the chase mode is activated during the time determined by its ρ_8 rule.

Algorithm 3.16 Items interaction

$food(\rho_6)$	`If(`$collide(food)$`)` $\rightarrow$
	`Do(`$update(points, points + 1)$`)`
$coins(\rho_7)$	`If(`$collide(coin)$`)` $\rightarrow$
	`Do(`$update(points, points + 10)$`)`
$chase(\rho_8)$	`If(`$compare(chase, true, equal)$`)` $\rightarrow$
	`If(`$timer(5)$`)` $\rightarrow$
	`Do(`$update(chase, false)$`)`

During that time frame, the Pac-Man actor can eliminate Ghost actors as it is determined at ρ_9 rule. Conversely, if it collides with a Ghost actor and the chase mode is not active, the Pac-Man actor is destroyed.

Algorithm 3.17 Enemy interaction

$enemyCollision(\rho_9)$	`If(`$collide(ghost)$`)` $\rightarrow$
	`If(`$compare(chase, true, equal)$`)` $\rightarrow$
	`Do(`$update(points, points + 50)$`)` $\wedge$
	$\neg$ `If(`$compare(chase, true, equal)$`)` $\rightarrow$
	`Do(`$delete(pacman)$`)`

Ghost

The Ghost actors behaviour is initialised by a constant speed and by their ρ_1 rule, which controls the direction of the actor after colliding with the maze walls. It is necessary to point out that the random function is actually just run once, but it is displayed twice to fulfil the positive and negative ways of the predicate If on the IF-THEN-ELSE rule structure.

Algorithm 3.18 Enemy interaction

$wallCollision(\rho_1)$	`If(`$collide(wall)$`)` $\rightarrow$
	`If(`$compare(random(-1, 1), 0, less)$`)` $\rightarrow$
	`Do(`$update(velocity.x, velocity.x \times -1)$`)` $\wedge$
	$\neg$ `If(`$compare(random(-1, 1), 0, less)$`)` $\rightarrow$
	`Do(`$update(velocity.x, velocity.x \times 1)$`)`

Also, they have the ρ_2 rule that controls if the Pac-Man actor is in chase mode; in that case, they must change their display image to a "scared" one.

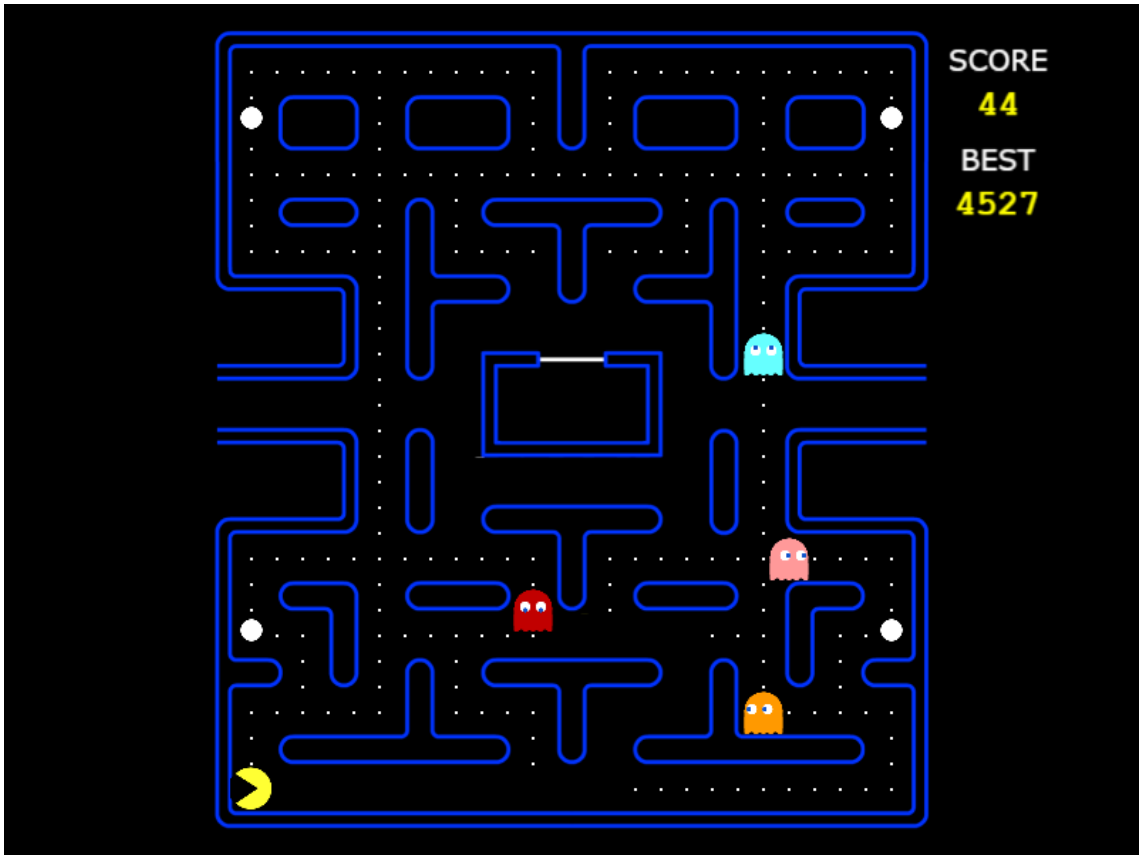

Figure 3.4: A capture of the Pac-Man game.

Algorithm 3.19 Enemy scared

$scared(\rho_2)$

$$\texttt{If}(compare(chase, true, equal)) \rightarrow$$
$$\quad \texttt{Do}(update(image, scaredImage)) \wedge$$
$$\neg\, \texttt{If}(compare(chase, true, equal)) \rightarrow$$
$$\quad \texttt{Do}(update(image, normalImage))$$

Similarly, in this case, if they collide with the Pac-Man actor, they are removed as it is indicated by their ρ_3 rule.

Algorithm 3.20 Enemy destroy

$destroy(\rho_3)$

$$\texttt{If}(collide(pacman)) \rightarrow$$
$$\quad \texttt{If}(compare(chase, true, equal)) \rightarrow$$
$$\quad\quad \texttt{Do}(delete(ghost))$$

The Pac-Man game has complex features that could be solved with this approach based on MAS. Both the Pac-Man actor and the Ghost actors have similar behaviours, the difference lies in the decision-making process for the direction change: the Pac-Man actor relies on the interaction with the game user while the Ghost actors decide to change direction arbitrarily after colliding with the maze walls. Simultaneously, the Ghost actors change their behaviour when the Pac-Man actor is in chase mode. This behaviour is derived from the perception of the Ghosts actors on the state of the Pac-Man actor. A capture of the game during its run is shown in Figure 3.4.

3.6.4 Other Games Developed

Independently from the use cases analysed in this section, a series of additional games have also been developed to support the validation of the system's competences. These games tried to test multiple video game genres and their respective game mechanics. The games under consideration are the following:

- **Arkanoid**. A classic puzzle game where the ball bounces until there are no bricks left.
- **Asteroids**. A classic space and third-person shooter where the asteroids are subdivided after being hit.
- **Groñof! Adventures**. A pointer-based game in which the player must eliminate enemies to move to the next level.
- **Ducks**. A first-person shooter game based on the classic ones found in fairgrounds.
- **Diamond Crush**. A match-three puzzle video game, in which the players have to swap diamonds on a game board to produce combinations of three or more with the same colour.
- **Fallas the Game**. A physics-based game, inspired by the well-known Angry Birds game, where the player has to burn the target to ashes.
- **Abstract Adventure**. A classic platform game where the goal is to elude enemies and traps at the same time as every coin in the scene is gathered.

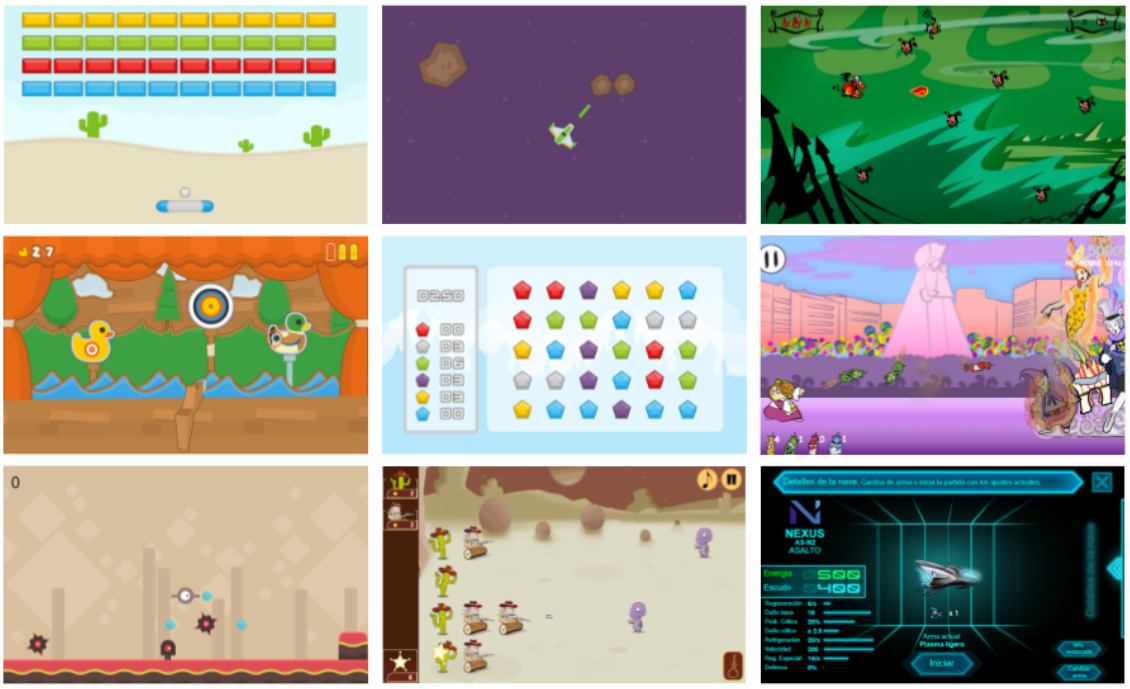

Figure 3.5: Captures of the games developed in the game engine.

- **Cowboys vs Aliens**. Tower defence game where the Cowboys defend their land from the Aliens.
- **Afterglow**. A horizontal scroll game, where the goal is to travel for as long as possible shooting and avoiding collisions with the enemies and their shoots.

In Figure 3.5, captures of these nine games during their run are presented in the same order of the previous list. Additionally, a set of these games presented in this section have been arranged on a web page [ML20].

3.7 Conclusions and Future Work

This work presents the definition of a game engine able to produce games that meet the requirements of a MAS. It draws from the formal definition of the MAS to conduct a description of the game and its essential elements. The game represents the environment and the actors are the agents of the multi-agent games generated by this game engine. From the environment state, the actors can perceive information and react to specific states based on their predefined tasks. The

behaviour associated with the tasks is determined by sets of behaviour rules and a pre-established logic semantics.

The construction of the game is based on a unique type of actor and without hierarchical relations between them. Each one of them has the same structure of properties and behaviour rules. All this without a scene graph, which simplifies the engine internal architecture. Moreover, the specification of the engine has reached a level of abstraction for the actor's behaviour definition that makes unnecessary complex data structures such as vectors or matrices during the creation of most arcade games.

Regarding the actor's behaviour specification, it has been proved that just five conditions and four actions are enough to define games using predicate formulas. It avoids the scripting for the creation of games, making that as one of the main features of the proposed engine. Also, all this is achieved without logical operators, matrices and loops, since the game cycle performs cyclical evaluations of the behaviour rules, just by using the IF-THEN-ELSE structure.

The use cases presented in this article show the potential of the game engine for the generation of games as MAS. Besides that, it has been observed that the system can generate logic, mechanics and complex behaviours from a very reduced first-order logic semantics as a behaviour definition descriptors.

As future work, the possibility of applying this methodology to the development of game engine architecture is proposed, in order to formalise an underdeveloped research field. Thus, the game engines design could be reformulated and the parallelisation of their behaviour could be explored.

Chapter 4

A Game Logic Specification Proposal for 2D Video Games

Publication

Marín-Lora, Carlos., Chover, Miguel., & Sotoca, Jose. M. (2020, April). A Game Logic Specification Proposal for 2D Video Games. In World Conference on Information Systems and Technologies (pp. 494-504). Springer, Cham. (Core C). https://doi.org/10.1007/978-3-030-45688-7_50.

Abstract

The game engines are one of the essential and daily used applications in game development. These applications are designed to assist in the creation of games' contents. However, the games definition and specification is still complex. The mechanisms involved in the behaviour definition are not easy to standardise, given their dependency with the subjective game logic design and the selected programming language features. In this sense, this work presents the design and development of a game logic system for 2D video games. It draws from the study and analysis of the behaviour and game logic definition literature. From this, a game logic system has been developed from first-order logic semantics and a reduced set of actions and conditions. The model has been tested with several games to determine its potential. Finally, one of these games is described with the proposed semantics, and further on it is also used as a reference for a user test against other game logic

systems.

Keywords

Game logic, Game development, First-order logic

4.1 Introduction

The development of video games is a complex process where multidisciplinary skills are required to complete it [Gre18, Nys14, Mil19]. The appealing of the games depends on the designer's skills to complete the game requirements, either for character design, animation, sound composition, narrative or game-play design. To address this problem, game engines provide the required tools to develop a game. In fact, professional game studios have their own one, although there is a trend towards the usage of commercial engines such as Unity [Uni22] or Unreal [Epi22]. However this fact has two issues, on the one hand, there is some obscurantism about the proprietary game engines, while on the other hand, the commercial engines implement a huge number of features turning them into really complex applications to use for the non-expert public. Additionally, from a theoretical point of view, there are claims to expand the game engines research field, since the formal requirements are unknown and there is little literature [AEMC08, AS10]. Specifically, Anderson et al. [AEMC08] highlights the lack of literature in this regard and proposes several research lines that should be explored in the future: a list of common software components, a unified language, generic content creation tools, a generic architecture and the identification of the best practices.

Conversely to the general trend towards the democratisation of content creation, it is still very difficult to work with game engines. Especially on the game logic definition, since they imply knowledge of scripting and game development APIs. The game logic definition is arguably the most subjective technical process in the video game development. In a script-based game engine such as Unity, two different programmers could solve the same mechanic differently depending on their initial approach. This is one of the reasons why the game logic standardisation is complicated, which hinders the game logic optimisation.

Currently, there are alternatives such as visual scripting systems. These systems transfer traditional programming to more comfortable visual elements for non-expert

users. Indeed, commercial game engines, as mighty as Unreal, use a Blueprints system based on message passing. However, these systems are still not easy to use and they still rely on huge APIs. It is delicate to determine which is the proper alternative to save these issues since the simplicity perception or ease of use brings forward subjective experience variables.

In this sense, this work proposes a 2D game logic specification along with a game engine requirements to run it. In order to verify its proper operation, a minimal game engine has been implemented meeting essential requirements. However, the game logic specification method proposed in this work could be implemented in any game engine. The proposal aims to generate a theoretical knowledge that serves as a reference for the generation and optimisation of game engines, providing knowledge on the essential requirements for the creation of 2D games. As a starting hypothesis, it is considered that it is possible to define complex behaviours from a reduced set of logical elements. During its development, a large number of 2D games have been implemented to check the capabilities of the system and, as far as it has been tested, the response has been positive since until now all the mechanics have been completed. As an example for this paper, a 2D arcade platform game is described down below. In addition, that game has also been used as a reference for a user test where its development is compared with two game engines and their game logic systems.

The rest of the article is organised as follows: Section 4.2 lists the game engine requirements, in Section 4.3 the game logic specification system is stated and in Section 4.4 the functions that encapsulate it are presented. Next, in Section 4.5, the 2D arcade game is described and then Section 4.6 presents the experiment carried out to determine if it is perceived as easier to use. Finally, Section 4.7 shows the conclusions gained from this development and the possible future work lines.

4.2 Game Engine Overview

Before detailing the game logic system, it is necessary to define some functional requirements for the minimal game engine. For simplicity, the game engine is conceived for 2D games and it must include the following features:

- The game is composed of a certain number of scenes.
- The game has a set of properties, available for the entire system.

- A scene is composed of a set of game objects with no hierarchies.
- A game object has different components: physics, sound, rendering, etc.
- Each component has a set of properties.
- Each game object has a set of behaviour rules.

Besides that, the game logic system requires a series of characteristics for its specification. These must allow game objects to support the game logic completeness, providing them with the following capabilities:

- Create and delete game objects.
- Read and update game properties or game objects properties.
- Execute changes and queries about any property.
- Check execution properties such as collisions, interaction events and timers.

In such a way, the system shall ensure a complete operation of the game state and its elements. Where the run of the game loop works with its properties. These properties define the game and its game objects features. In the proposed game engine, a set of generic properties have been selected for the game and the game objects, and both can handle new knowledge as new properties.

Game

- **General**: Name, resolution, active scene, and camera properties.
- **Sound**: Sound files, start, volume, pan and loop.
- **Physics**: Gravity and air friction.
- **New**: Custom shared properties for the game, admitting boolean or numeric.

Game Object

- **General**: Name, position, angle, dimensions, collision shape and tags.
- **Render**: Texture, opacity, flip, scroll and tile.

- **Text**: Text, font, size, color, style and offset.
- **Sound**: Sound files, start, volume, pan and loop.
- **Physics**: Velocity, rotation, density, friction, restitution and damping.
- **New**: As for the game, custom properties to complete functionalities.

4.3 Game Logic Specification

The game logic assigns to the game objects the tasks to perform based on the game state, following a predefined semantics. Its development is based on first-order logic (FOL), widely used in other areas such as robotics and automation [Gel03, PS85, WJK08]. The semantics are defined by a syntax of logical and non-logical symbols. Logical symbols include logical connectives and variables while non-logical symbols include predicates and functions [BLR92]. The domain *D* for this interpretation *I* is the set of the game objects and the game. The predicates Ψ, defined in FOL, specify the behaviour rules. In such a way, the tasks that a game object must perform are organised in predicate formulas

$$\Psi_0 \wedge \Psi_1 \wedge \Psi_2 \wedge \cdots$$

where its elements can have the following structures based on just two predicates: the *If* condition and the *Do* action.

- **Action structure**: Composed by an atomic element that includes a single literal predicate *Do*.
- **Conditional structure**: Modelled by a predicate sequence based on the IF-THEN-ELSE rule structure [Kar88] where *If* is a conditional literal, and both ϕ and θ are new predicate sequences to be evaluated if the condition is met or if it is not, respectively.

$$(\text{If} \rightarrow \phi) \wedge (\neg \text{If} \rightarrow \theta)$$

For the predicates, expressions are essential since they evaluate the game state based on parameters, either as a boolean or as an arithmetic expressions.

- **Arithmetic expression**: Performs mathematical operations either with numerical constants, game or game objects properties, or mathematical functions such as *sin*, *cos*, *tan*, *asin*, *acos*, *atan*, *sqrt*, *random* and *abs*.
- **Boolean expression**: Evaluates logical relationships between game elements and arithmetic expressions using relational operators such as *greater*, *greater-or-equal*, *equal*, *less-or-equal* and *less*.

Action Predicate *Do*

An action is defined as a behaviour to be performed by a game object. In this specification, the actions are formalised as the non-logical function elements which can handle parameters as arithmetic expressions. Where the set of actions is based on the *create*, *read*, *update* and *delete* (CRUD) operations for information persistence on databases [Dai10] applied to game and game objects properties. From these operations, the system can produce more complex actions raising the system's level of abstraction. In this way, the available operations are:

- **Create**: Creates a new element, as a copy of an existing game object.
- **Read**: It allows to read the information of a game or a game object property. The syntax *gameObject.property* is used to read this information.
- **Update**: Modifies the value of a game or a game object property. The new value is determined from the evaluation of an arithmetic expression.
- **Delete**: Removes a game object from the game.

Conditional Predicate *If*

The conditional predicate represents the evaluation element for a condition in this decision-making process. Where the condition evaluation is based on the result of a boolean expression that assesses the relationship between properties.

$$If(booleanExpression)$$

Conceptually, this boolean expression evaluates the relationship between properties with an arithmetic expression. For this, it has an established structure composed of a read operation, a relational operator and an arithmetic expression.

$$property\ [\geq, >, ==, <, \leq]\ arithmeticExpression$$

Following this theoretical framework, the proposed game logic specification is able to define behaviours for 2D games.

4.4 Functions

From the previous section, the game logic system could compose game mechanics. However, this low-level semantics is not the most comfortable to use. This section presents the function encapsulation of the low-level operations presented above aiming to increase the level of abstraction without giving up on its potential. For this purpose, a reduced set of actions and conditions are defined, in order to create high-level functions to ease the behaviours definition.

Actions

- **Spawn**: Directly derived from the *create* operation, it spawns a game object in a position and in an angle as a copy of an existing one. This action is normally used to create enemies or projectiles.
- **Edit**: Execution of an *update* operation to modify a property from the game or a game object. The value comes from an arithmetic expression evaluation.
- **Destroy**: Implements the *delete* operation over a game object.
- **Move**: Specialisation of the *update* operation that applies a displacement on the game object by an angle and a speed as arithmetic expressions.
- **Rotate**: Equivalent to the *Move* action for angular displacements depending on a pivot and a speed, where both parameters are arithmetic expressions.
- **Push**: Physics-based alternative to the *Move*, where a force is applied to the game object in a given angle. It relies on the object's physics component.
- **Animate**: Texture swapping to produce a key-frame animation controlled by an arithmetic expression for the frames-per-second rate.
- **Play Sound**: Audio playback of a sound stored in the game data.
- **Change Scene**: Scene swapping to change the information from a scene to another. This action could be used to switch between game levels.

Conditions

- **Check**: Evaluation of a boolean property from the game or a game object.
- **Compare**: Relational condition between a property with an arithmetic expressions in modes *greater*, *greater-equal*, *equal*, *less-equal* or *less*.
- **Timer**: Temporary condition for a specific time where a system timer is compared with a cut-off time. It will always be false as long as the timer is lower than the cut-off time. When true, the timer needs to be restarted.
- **Collision**: Collision detection between game objects based on tags.
- **Keyboard**: Keystrokes condition for predetermined keys in a key mode.
- **Pointer**: Pointer controller managed by the game engine's input system.

4.5 Use Case

During the development of this work, several 2D games have been implemented in order to test its performance and its capabilities. In this section, a 2D platformer inspired on the *London's Tower Bridge Defense* game [Uni19] is presented as an example. The selection of this game is underpinned by the reliability of its design, since it is one of the advanced 2D tutorial games of the Unity learning platform and it is assumed that its mechanics and procedures are optimised and well structured. Figure 4.1 shows a capture of the outcoming game in its execution.

For copyright reasons, the assets employed are from a public library with no usage restrictions [Ken19]. Which has just affected the game's aesthetics aspect. Next, the game is described by the semantics established throughout this work.

Game Objects

- **Player**: The game's character controlled by the user. It has to accumulate as many points as possible by destroying *Enemy* game objects while avoids their contact. It can move to the right, to the left, jump and shoot *Bullet* game objects over the enemies. Points and lives are stored in custom properties.
- **Enemy**: The antagonist character who falls into the scene and randomly chooses a movement direction. It bounces when collides with the scene

Figure 4.1: Capture of the 2D platformer game.

horizontal limits until it falls through the drain and is eliminated. It can harm the *Player* and can be destroyed with *Bullet* game objects.

- **Ground**: Static object setting the *Player* and the *Enemy* walkable zones.
- **Bullet**: Spawned by the *Player* to destroy *Enemy*s. It moves in the *Player*'s direction until it reaches an *Enemy* or the scene limits, then it is destroyed.
- **Aid**: *Player*'s health recovery, randomly spawned by the *Aid Spawner*. When they interact, the *Player*'s lives are increased and it is destroyed.
- **Score**: Scoring display for the *Player*'s points property. It is increased if a *Enemy* is destroyed by a *Bullet*.
- **Stopwatch**: Time controller for the sixty seconds available for the game. When it reaches zero, the game ends.
- **Enemy Spawner**: Auxiliary object who randomly drops *Enemy* objects from the screen top.
- **Aid Spawner**: *Enemy Spawner* akin, but dropping *Aid* in a lesser frequency.

Each game object has a behaviour rules set to perform their tasks. Since present all the behaviour rules is unrealistic given its extension, the rules associated with the *Player* and the *Enemy* movements are described next. For its correct operation, some of their properties have been balanced. Generally, in a platform game, the *Player* is restricted to shifts from left to right and to jumps. In this case, the jumps are constrained by a ground-contact condition. Additionally, the user interaction works with keyboard events, but the same structure would be valid with pointer events.

Algorithm 4.1 Player movement to the left

```
If(keyboard(keyLeft, pressed)) →
    If(compare(maxSpeed, greater, abs(player.velocityX))) →
        If(collision(groundTag)) →
            Do(push(player.thrust, 180)) ∧
            Do(edit(player.horizontalFlip, true)) ∧
            Do(animate(player.walkAnimationTextureList, 24))
```

Algorithm 4.2 Player movement to the right

```
If(keyboard(keyRight, pressed)) →
    If(compare(maxSpeed, greater, abs(player.velocityX))) →
        If(collision(groundTag)) →
            Do(push(player.thrust, 0)) ∧
            Do(edit(player.horizontalFlip, false)) ∧
            Do(animate(player.walkAnimationTextureList, 24))
```

Algorithm 4.3 Player jump

```
If(keyboard(keySpace, pressed)) →
    If(collision(groundTag)) →
        Do(push(10 × player.thrust, 90)) ∧
        Do(animate(player.jumpAnimationTextureList, 24))
```

Conversely, the *Enemy* movement is completely different since its movements are autonomous and are determined by two behaviour rules dependent on collision events. The first happens with its first contact with the central ground, when it will randomly select a movement direction (left or right). The second is triggered if it reaches the scene horizontal limits, when it will shift its direction.

Algorithm 4.4 Enemy movement initialization

```
If(collision(centralPlatformTag)) →
    If(check(enemy.first)) →
        Do(edit(enemy.velocityX, thrust × (12 × random(0, 1)))) ∧
        Do(edit(enemy.first, false))
```

Algorithm 4.5 Enemy boundaries

```
If(collision(boundariesTag)) →
    Do(edit(enemy.velocityX, enemy.velocityX × 1))
```

4.6 Experiment

From the previous section, an experiment has been carried out in order to check if it works in an easier way than other methods. During the activity, the participants developed the *Player* movement (left and right shifts, and jumps) in Unity, Unreal and in a minimal game engine known as GLS arised from this work. For Unity, the scripting system was C# while for Unreal was its Blueprints system. Besides that, the GLS method relies on a decision tree visual editor based on the Section 4.4's theory. Upon completion, the participants filled a usability test in order to check if the proposed system is perceived as easier to use than the other commercial methods. It draws from the hypothesis that the game logic specification can be stated in a easier way than other traditional methods. The method selected to measure the degree of usability perceived is the System Usability Scale (SUS) [B+96]. This method provides a quick metric through a 10-item questionnaire with five response options, from the most absolute disagreement (1) to the most absolute agreement (5). The questionnaire items are the following:

Q1. I think that I would like to use this system frequently.

Q2. I found the system unnecessarily complex.

Q3. I thought the system was easy to use.

Q4. I think that I would need the support of a technical person to be able to use this system.

Q5. I found the various functions in this system were well integrated.

Q6. I thought there was too much inconsistency in this system.

Q7. I would imagine that most people would learn to use this system quickly.

Q8. I found the system very cumbersome to use.

Q9. I felt very confident using the system.

Q10. I needed to learn a lot of things before I could get going with this system.

4.6.1 Results

The experiment was carried out with 27 participants. Three of them had never used any of the evaluated systems, while the rest knew at least one and only two of them had worked with the three systems. In order to measure the users' previous experience, they have rated their previous knowledge in a range from 1 to 5, from nonexistent to expert respectively, where the average result obtained was 2.85. Besides that, game comprehension, a practical demonstration was carried out in all three cases with the *Player*'s movement specification. The results obtained are shown in Table 4.1. The three central columns show the results average obtained by the methods in each item of the questionnaire along with its standard deviation shown in parentheses. In order to check the questionnaire statistical significance, a Friedman's non-parametric test has been applied [Fri37]. This is a non-parametric technique to measure the significance of the statistical difference between the applied methods. In this case, the Friedman test considers all ratings with the same importance and to discern the statistical significance of the results the Chi-square distribution was used. For all three methods, the Chi-square value is 5.99 with a confidence level of p = 0.05, which would imply that the resulting values lesser than this reference value do not have statistical significance. The values of the statistical significance are shown in the right column of Table 4.1, where the positive statistical significance is indicated by (+) when the value is greater than 5.99, and by (-) in the opposite case.

4.7 Conclusions and Future Work

This work proposes a game logic specification to define 2D games mechanics based on a reduced set of behaviour elements. The analysis carried out on the

Table 4.1: SUS test results and Friedman significance evaluation

Items	Unreal	GLS	Unity	Friedman
Q1	3.04 (±0.98)	3.96 (±1.13)	3.04 (±1.43)	10.9 (+)
Q2	3.52 (±1.25)	1.81 (±1.04)	3.93 (±0.96)	24.5 (+)
Q3	2.67 (±1.21)	4.56 (±1.09)	2.41 (±1.39)	22.2 (+)
Q4	3.81 (±1.14)	1.70 (±0.82)	4.00 (±1.18)	32.1 (+)
Q5	3.11 (±1.15)	4.22 (±1.01)	2.93 (±1.33)	9.9 (+)
Q6	3.04 (±1.06)	1.93 (±0.87)	3.37 (±1.60)	9.2 (+)
Q7	2.67 (±1.18)	4.81 (±0.48)	2.04 (±1.06)	35.6 (+)
Q8	3.85 (±0.99)	1.37 (±0.63)	3.89 (±1.22)	33.9 (+)
Q9	2.25 (±0.94)	4.44 (±0.70)	2.59 (±1.45)	25.7 (+)
Q10	3.93 (±0.96)	1.59 (±0.75)	4.26 (±0.98)	38.7 (+)

requirements and on game engines features have shown that there is a research need to determine the games' essential requirements and the correct procedures for their design. From this information, a data model has been built supporting the functional requirements raised at the beginning of the paper. Besides that, a semantics based on first-order logic has been defined with the aim of generating logic rules from a reduced set of actions and conditions. All this without hierarchies, loops or complex data structures such as matrices or vectors. Additionally, the proposed system has been tested against two other commercial systems such as Unity and Unreal. The first one based on traditional scripting and the second one based on message passing. The results show that the users perceived GLS as more usable than the other systems analyzed. The Unity and Unreal methods were virtually tied for second place, although there is a certain advantage in favour of Unity, probably due to some users who were previously experienced on this one. This fact is stated in Q1, where GLS beats Unity and Unreal in preference of use with a difference of almost one point. Nevertheless, this has to be carefully considered due to its relationship with subjective preferences, motivations and previous experience. The questions Q2 and Q3 show that GLS is perceived as easier to use against the other two. In addition, it seems that Unity's scripting language is a bit more complex for the users than a message-passing language as the Unreal Blueprints. In this way, it follows that users establish as unnecessary the use of complex methods to complete these game development tasks. In fact, question Q4 supports this conclusion and confirms that it is necessary to reduce learning curves in order to democratise game development. On questions Q5 and Q6 regarding the integration

of functions and the inconsistencies detected, the results show a clear advantage of GLS. Since it uses a reduced set of actions and conditions while Unreal and Unity methods are penalised for the large number of functions they contain and their huge amount of parameters. In addition, the users considered GLS as much faster to learn than the others systems and that they also find it much less cumbersome to use than Unity and Unreal, with a difference of almost two points in questions Q7 and Q8. Finally, Q9 and Q10 show that GLS seems safer for upstarters, again a two points difference in their averages. As future work, it is intended to keep exploring the capabilities of this design in three ways: its extension to a 3D game engine, its application on a multi-user point of view and its potential implementations in parallel.

Chapter 5

A First Step to Specify Arcade Games as Multi-agent Systems

Publication

Marín-Lora, Carlos., Cercós, Alejandro., Chover, Miguel., & Sotoca, Jose. M. (2020, April). A First Step to Specify Arcade Games as Multi-agent Systems. In World Conference on Information Systems and Technologies (pp. 369-379). Springer, Cham. (Core C).
https://doi.org/10.1007/978-3-030-45688-7_38.

Abstract

The lack of formalities in the development of video games is one of the main obstacles for the incorporation of new professionals to the field. Although there are general proposals to describe and specify video games with techniques such as Game Design Document or Game Description Language, these are usually aimed at implementations in predetermined media, which determines the game specification from the outset to its implementation in the selected platform. This paper proposes a method for the definition, specification and implementation of a video game based on multi-agent systems, where its elements, functionalities and interactions are established independently of the platform used for its development. To prove its validity and capabilities, the classic arcade game Frogger has been used as a demonstrator. This game has been defined in its general form and subsequently

implemented on three different platforms following the same specification. Each of these implementations has been made using different engines, languages and programming techniques, but in any case, meeting the requirements of the game and multi-agent systems.

Keywords

Computer games, Game specification, Multi-agent systems

5.1 Introduction

Despite being an industry with more than forty years of existence, the design and development of video games lack formalities and standards that unify the definition and specification process of its products. In this sense, the design and development of video games still has several open fronts demanding to keep researching in order to achieve optimum performance in the definition and specification, development and implementation processes. Actually, some authors indicate that it would be necessary to define a formal language that standardises concepts and processes present in any development [AEMC08, AS10]. In fact, this absence of formalities makes the initiation in the design and development of games for non-expert people a truly complicated task. In order to find new models to specify video games formally, it has been recognised that multi-agent systems (MAS) have several similarities with video games. They are systems composed of entities, called agents, living and interacting with each other in a shared environment, where each agent has a set of properties and a set of action rules to determine their tasks and goals [Woo09]. These systems are widely used in fields such as control systems or artificial intelligence to solve complex problems, by dividing complex tasks into simpler ones and assigning them to agents. The analogy between the definition of a MAS and the specification of a video game seems apparent: the agents play the role of game elements, usually known as game objects; interacting with each other and sending information about the game logic, the collisions between them or the user interaction, and where each one has a task to fulfil in the game based on a set of behaviour rules.

This paper presents a specification model based on MAS to prototype, define and specify games regardless of the platform on which it is intended to be implemented.

In order to demonstrate the validity of the model, the definition of a classic arcade game with frequent features and mechanics is proposed along with its implementation in three different platforms. The reason for choosing a multi-agent model is because the behaviours and interactions in these systems have correspondences with the behaviours and interactions between individuals and their environment in the real world [FW99]. For this reason, it has been considered that the first steps for non-expert people are more affordable by taking the game as an analogy with the real world [RMLRC18, MLCSG20]. In this sense, this proposal is based on the hypothesis that MAS are suitable for the specification and implementation of video games. In order to test this hypothesis, the study is focused on arcade games because their mechanics have been replicated and extended consistently since the first video games. Specifically, the analysis draws from the description of the Frogger game [KS81] and its implementation in NetLogo [TW04], a programming environment based on turtle language oriented to the MAS prototyping. Based on that model, the game has been implemented in Gamesonomy a visual programming 2D game engine [Gam21] and in the Unity [Uni22] game engine. Subsequently, an analysis of the differences and particularities of each case has been composed.

Finally, the document presented is organised as follows: Section 5.2 presents the state of the art related to software and video game specification systems using MAS. Next, Section 5.3 details the model that has been followed to formally define the analogy between video games and multi-agent systems. After that, in Section 5.4, the use case is presented with which the validity of the model in different game engines will be analysed, together with the results obtained after that experience in Section 5.5. Finally, in Section 5.6 several conclusions from the realisation of this study are presented, and its possible future derivations.

5.2 State of the Art

The biggest obstacles faced by upstarts in the development of video games appear at such early stages as the design and specification of a project. In other words, in the process of translating the creative process into requirements lists and tasks specification, as it would be made in software engineering. Over the years, this problem has been tried to be addressed with tools such as the "Game Design Document" (GDD) models, where the creative process must be described and mapped into a breakdown of requirements, tasks and goals [CNS05]. However,

there is no standard GDD format to describe the game mechanics and no technical reference language that standardises technical concepts [AEMC08], which usually causes gaps between the established creative concept and the software developed [AS10]. In fact, one of the first problems to address when starting a project arises with the platform selection, taking into account its capabilities and limitations to correctly define the game. Some studies present models to determine which is the appropriate platform for a project [AMW+13], while others look for methods to reuse video game specifications between different platforms extracting common features from a several game development platforms [BMR07]. From a research standpoint, the quest for a generic framework has been present in the video game literature for a while. For instance, and despite doing so indirectly, the formal definition of games has received contributions from the AI research for games and from competitions in General Game Playing (GGP) [GLP05, PLST+16, Thi10, Thi11] where the same system must learn to play any game based on a Game Description Language (GDL) [LHH+08, ELL+13]. In the grey area between traditional methods as GDD and others as advanced as GDL, some papers have already studied the possible synergies between the concepts of GDL and MAS [ST09]. The MAS are composed of sets of agents that interact and make decisions with each other within a shared environment [Woo09, FW99, GQCC10]. Where behaviours have traditionally been defined by decision theory and game theory metrics [PW02]. Among the uses of MAS, on robotics and autonomous systems they have satisfied critical real-time restrictions [OD98, BCJS99], also the implementation of virtual commerce and trade fairs [RGR+15, GQC+07b], obstacles avoidance in navigation [VdBLM08] and mailing delivery using mobile robots [CBJ+08]. In the video games field, they have been used to simulate large number of people in restricted areas [ATE+12] or to prototype the development of game engines [MLCS19].

5.3 Video Games as Multi-agent Systems

The goal of this work is to define a methodology for the specification and implementation of video games. The proposal focuses on the analogy between MAS and video games. The MAS have formal specification aspects to define a video game in a generic way, integrating specific aspects of the game such as the game logic with its entities and its behaviour rules, or the game physics with the detection and response of collisions between game elements. The method for the definition

of a game must consider the features of the game, the elements that compose it, the behaviours definition and the user interaction. From this point, it is necessary to make a formal analogy between these game concepts and their correspondent concepts in a MAS.

Inside the shared environment, each agent has sets of properties and behaviour rules determining their interaction with each other. In order to prove that a game can be expressed as a MAS, a formal theoretical framework is presented. For this purpose, the definition of agent proposed by M. Wooldridge [Woo09] has been used, where an agent is a computer system located in an environment and is capable of performing tasks autonomously to meet its design objectives. In addition, this method is also based on the work done by Marín-Lora et al. [MLCSG20], where a game engine is defined as MAS. As a summary, some of the characteristics of the theoretical framework used for this proposal are detailed below:

- The environment to which the agents belong can be in any of the discrete states of a finite set of states E = $[e_0, e_1, e_2, \cdots]$.
- The environment shared by all agents has a set of properties that determine their status and that can be accessed by any agent in the environment.
- The system agents have a series of generic properties common to all of them: geometric, visual, physical, etc. In addition, they have the ability to assimilate new properties to perform specific tasks.
- Agents have behaviour rules to alter the environment state or an agent state in order to fulfil their plans and objectives.
- The agents have a set of possible actions available with the ability to transform their environment Ac = $[\alpha_0, \alpha_1, \alpha_2, \cdots]$.
- The run r of an agent on its environment is the interlayered sequence of actions and environment states $r : e_0 \rightarrow^{\alpha_0} e_1 \rightarrow^{\alpha_1} e_2 \rightarrow^{\alpha_2} \cdots e_{u-1} \rightarrow^{\alpha_{u-1}} e_u$.
- The set of all possible runs is R, where R^{AC} represents the subset of R that ends with an action, and R^E represents the subset of R that ends with a state of the environment. The members of R are represented as R = $[r_0, r_1, \cdots]$.
- The state transformation function introduces the effect of the actions of an agent on an environment τ: $R^{AC} \rightarrow \wp(E)$ [FHMV04].

In order to transfer these concepts to a video game, it is necessary to implement the general characteristics of the game and the objects of the game such as those of an environment and its agents, respectively. Taking into account an essential requirement for the validity of this work: the same functions and attributes must exist for each element, regardless of the limitations or characteristics of the game engine or software environment selected for its implementation. This proposal follows the methodology presented in Marín-Lora et al. [MLCSG20], where the rule specification is structured using a first-order logic semantics [BLR92] based on two predicates: a condition *If()* and an action *Do()*. Where each predicate can run calls to other actions α of the system, or evaluate arithmetic and boolean expressions. An example of these rules and the technical specification of the game could be presented in the following fashion:

Environment

Assets: { *Sprites, Sounds* }

Properties: { *General, Physics, Input, Logic, Audio, Render* }

Agents

Agent1:

- **Properties**: { *propertyA: true, propertyB: 7.52, propertyC: "Hello"* }
- **Scripts**: { script1: { *If(A)* $\rightarrow$ *Do(B)* }, script2: { *Do(C)* } }

Agent 2:

- **Properties**: { *propertyA: false, propertyB: -5, propertyC: "Hi again!"* }
- **Scripts**: { script1: { *If(A>B)* $\rightarrow$ *Do(C)* }, script2: { *If(D)* $\rightarrow$ *Do(E=E+15)* } }

5.4 Use Case: Frogger

In order to determine if a game can be implemented in the same way on different game engines based on this proposal, the Frogger is going to be described following a MAS specification. First, the concept and objectives of the game are described. Subsequently, the game specification will be presented following the methodology outlined in the previous section. This game was selected because it implements

several arcade games mechanics and there is an open-source implementation in NetLogo, which puts in context and highlights further issues.

The Frogger game presents a scene where the player drives a frog passing through a road crossed by cars and trucks, and a river with trunks and turtles until reaching a safe area of water lilies. Initially, the frog is safely located at the bottom of the screen and can move up, down, right or left. At the beginning of the game, the frog has 5 lives, which can lose if collides with a car, a truck, or if it contacts with the water. Game elements representing cars and trucks come up from the right side of the screen and move with constant speed to the left side. To overcome the road, the frog must not contact with a car or a truck. For the water, it needs to remain on a trunk or turtle noticing that turtles can sink during arbitrary intervals. The player will win the level when it reaches the water lilies at the top screen. The next level would start-up to a maximum of 5.

Environment

Assets: { *frogSprite, truckSprite, carSprite, trunkSprite, turtleSprite, waterlilySprite, jumpSound, collisionSound, drownSound, winSound* }

Properties: { *width: 20, height: 20, centerX: 0, centerY: 0, lives: 5, levelComplete: false, currentLevel: 0, stopwatch: 0, timeLimit: 60, diveProbability: 0.1* }

Agents

Game Manager:

- **Properties**: { }
- **Scripts**: {
  * LevelComplete: { *If(Env.levelComplete)* $\rightarrow$ *(If(Env.currentLevel < 4)* $\rightarrow$ *Do(Env.currentLevel = Env.currentLevel+1)* $\wedge$ *Do(resetLevel))* }
  * ResetLevel: { *If(Env.lives* $\neq$ *Frog.lives)* $\rightarrow$ *Do(Env.lives = Frog.lives)* $\wedge$ *Do(resetLevel)*}
  * Stopwatch: { *Do(Env.stopwatch = Env.stopwatch+1)* $\wedge$ *If(Env.stopwatch* $\geq$ *Env.timeLimit)* $\rightarrow$ *Do(Frog.lives = Frog.lives-1)* $\wedge$ *Do(Env.stopwatch = Env.timeLimit)* }
  * EndGame: { *If(Env.lives == 0)* $\rightarrow$ *Do(endGame)* }

  }

Frog:

- **Properties**: { *x:0, y:Env.centerY-Env.height/2, lives:5, jumpStep:1, jumpCount:0* }
- **Scripts**: {
  * LeftJump: { *If(Env.leftKey) → Do(x = x - jumpStep) ∧ Do(rotation = 0) ∧ Do(jumpCount = jumpCount + 1)* }
  * RightJump: { *If(Env.rightKey) → Do(x = x + jumpStep) ∧ Do(rotation = 90) ∧ Do(jumpCount = jumpCount + 1)* }
  * UpJump: { *If(Env.upKey) → Do(y = y - jumpStep) ∧ Do(rotation = 180) ∧ Do(jumpCount = jumpCount + 1)* }
  * DownJump: { *If(Env.downKey) → Do(y = y + jumpStep) ∧ Do(rotation = 270) ∧ Do(jumpCount = jumpCount + 1)* }
  * VehicleCollision: { *If(vehicleCollision) → Do(lives = lives - 1)* }
  * Boating: { *If(boatCollision) → Do(x = x + boat.velocityX * boat.directionX)* }
  * WaterCollision: { *If(waterCollision) → Do(lives = lives - 1)* }
  * WaterLily: { *If(waterlilyCollision) → Do(Env.levelComplete = true)* }

  }

Car, Truck, Trunk:

- **Properties**: { *originX: Env.centerX + Env.width / 2, x: originX, y: 0, velocityX: 1, velocityY: 0, directionX: -1, directionY: 0* }
- **Scripts**: {
  * Movement: { *Do(x = x + velocityX * directionX)* }
  * Limits: { *If(x < -originX) → Do(x = originX)* }
  * Movement: { *Do(x = x + velocityX * directionX)* }

  }

Turtle:

- **Properties**: { *originX: Env.centerX + Env.width / 2, x: originX, y: 0, velocityX: 1, velocityY: 0, directionX: -1, directionY: 0, dive: false* }
- **Scripts**: {
  * Movement: { *Do(x = x + velocityX * directionX)* }
  * Limits: { *If(x < -originX) → Do(x = originX)* }
  * Dive: { *If (random(0 , 1) < Env.diveProbabilty) → Do(dive = true)* }

  }

5.5 Results

From the previous section, a version of the Frogger game has been implemented in three different systems: the MAS prototyping environment NetLogo and in the game engines Gamesonomy and Unity. All following the same guidelines: the game description and the game definition as MAS, but taking into account the differences and limitations of each system to achieve an identical result.

5.5.1 NetLogo

First of all, it is necessary to point out that NetLogo is not a graphical programming environment. Graphics and user interaction are not ideal to generate a good user experience. Examples of these limitations are the grid-based layout and the looping *ticks*, from which the drawing and logic must be managed explicitly. However, it allows to edit the interface, the scene and the frame-rate without using code: creating buttons that run functions or edit properties using sliders. The scene configuration is limited to the origin of coordinates and the scene dimensions. Also, the vertical and/or horizontal limits have to be determined, for an agent to exit at one end and appear at the other while moving.

For the game interface, four buttons have been arranged and linked to keyboard events and the movement functions, a native function *setup* callback to reload the level, a play function with the native *go* function and three sliders for the frog's lives in a range of 1 to 5, the initial level selection and the highest time available to complete the level from 60 to 10 seconds. Also, four monitors have been arranged that show the remaining lives, the current level, the time left and a jump counter. At the beginning of each level, each agent is created in its initial position by executing the *setup* function. Agents are generated by creating NetLogo agent objects called *turtle*, but for instantiation, this process must be performed with the *breed* function. Also, the game global variables must be defined. In the agent's case, each one except the frog has two new properties: speed and time. The speed determines the time interval that elapses between each jump. This action is controlled by a time variable, which counts the *ticks* remaining until the next iteration. Besides that, it is also arbitrarily determined that turtles can dive and which not by initialising their dive variable. Once in execution, the activation of the start button starts the game loop as long as lives is greater than zero. First, the game cycle records

user input events to determine the movement the frog must perform. It should be noted that the configuration of the movement of the frog, unlike the other agents, prevents it from crossing the horizontal limits of the scene. The implementation of the behaviour rules of the agents, it is possible to execute the actions of each agent when a predetermined condition is fulfilled using the native *ask* function. The agent's movement relies on a time property that decreases every *tick* until it reaches 0. When this condition is met, agents can move. The movement is normally performed by the forward function. However, trunk and turtle agents have special functions, since they must interact with the frog agent and in the case of turtles, they can be submerged in the river arbitrarily. The last step of the loop is to check if the frog has reached a waterlily and therefore has exceeded the level, or if it has collided with a vehicle or with the water and therefore has not exceeded the level. For the first case, the water lily texture would be changed to the frog one and the current level would be increased if possible. In the second case, one life would be subtracted and, if there are lives left, the level would be restarted.

5.5.2 Gamesonomy

Gamesonomy is a web-based 2D game engine to create games without using code. In this platform, the game objects are known as Actors, and their behaviours are defined by decision trees composed by a reduced set of conditions and actions. The game loop evaluates the rules in a continuous cycle. At the beginning of the loop, it is checked if the conditions are met and if they are, the evaluation flow continues its way across the proper tree branch until it reaches an action leaf.

The screen size and resolution are adjusted from the game properties, where global game properties can also be created such as the current level, lives, and stopwatch. It should be noted that Gamesonomy does not has a default option for board limits, so it must be determined for each agent if necessary. However, the game global functions need to be implemented in an actor. In this case, a game manager actor was created to run game behaviour rules. It is located in the middle of the environment, from where the level elements are instantiated. Among them, the setup rule is only executed at the game start. Also, a rule to remove all objects from the scene has been included. This function is activated for an iteration so that each agent in the scene could be self-destructed. In order to arrange the actors, first, it is necessary to create the instantiable ones (vehicles and boats) to be spawned in their

proper line. Conversely, the frog actor, which is neither instantiated nor eliminated. Furthermore, each actor has labels that identify them in their interactions. The frog has three rules: one checks the user's inputs to control its movement and the other two check if it collides with a vehicle or water and, if it does, it asks for the restart of the game. In the water case, if it is not in contact with a turtle or trunk actor. The other actors have rules that allow them to move, exit the screen and self-destruct if the game is restarted. For the movement, the timers have been used to control their speed so that from time to time it advances in the direction in which it is looking. Trunks and turtles can move the frog if they are in contact with it, and the latter can also dive in randomly if its dive property is true. The amount of turtles diving is controlled by a diving probability property, which can be set up in the game.

5.5.3 Unity

Unity is a general-purpose game engine that allows developing 2D and 3D games. In this case, the agent element is called game object. It is the basic component from which the scenes are composed, and it can store components to perform specific tasks. The behaviours are defined on the game objects with C# scripts. In this development, there is also a game manager storing the global properties and handling the state of the game and the game functions. A start function is included in its initialisation script, known as setup in the previous implementations, which is executed on the game load to initialise the local properties. The different types of game objects have been stored as "prefabs" so that the manager can instantiate them at the beginning of the game by accessing the reference of each prefab. The agents' movement function has been generalised so that it is compatible with all of them, also allowing them to access the customised velocity and direction properties. As mentioned previously, the movement function advances a unit and, in the case of turtles and trunks, drags the frog with the same amount of movement. The velocity value depends on the time differential elapsed since the previous loop, the interval between two consecutive executions. In this way, the speed can be measured in exact seconds. In the case of the truck and turtle agents, it has been necessary to create two scripts for specific behaviours: in the truck, the joint script coordinates the movements of the front and rear when leaving the screen, when the front leaves, it appears on the other side but the back no. In the turtle, the diving script is used to determine when a turtle that can dive is done and when it comes out, the turtles that

can dive are determined by creating the level in the manager. In the water lilies case, their only function is to check if the frog collides and notify the manager as soon as it does. Finally, the frog agent collects the user's inputs to move on the board and checks the collisions with other agents, calling the game manager if necessary.

5.6 Conclusions and Future Work

This paper proposes a method for the definition, specification and implementation of video games based on MAS, where its elements, functionalities and interactions are established regardless of the platform used for its development. To prove its validity and capabilities, the classic arcade game Frogger has been used as a demonstrator. This game has been defined in a general form and then implemented on three platforms following the same specification. Each of these implementations has been made using different engines, languages and programming techniques, but meeting the requirements of the game and the MAS.

The purpose of this work was to demonstrate that video games and MAS share several features and it is possible to improve video game development processes in this way. From the previous sections, it is extracted that the MAS features fit as a starting point for the video games definition and therefore it is necessary to keep working in methods that potentiate a symbiosis between these concepts. Additionally, the incorporation of agent specification systems into the video game development can ease the understanding of games before they are even implemented, which could ease the access into the sector of professionals who do not have technical experience creating video games.

As future work, it is intended to explore the potential of this method by designing and implementing a MAS-based game engine and generate games that meet the characteristics of the MAS. Trying to explore the capabilities of these interactive systems in games and in virtual and augmented reality experiences, and to provide game development tools to non-expert profiles such as children.

Chapter 6

A Multi-agent Specification for the Tetris Game

Publication

Marín-Lora, Carlos., Chover, Miguel., & Sotoca, Jose M. (2021, October). A Multi-agent Specification for the Tetris Game. In International Symposium on Distributed Computing and Artificial Intelligence (pp. 169-178). Springer, Cham. (Core C). https://doi.org/10.1007/978-3-030-86261-9_17.

Abstract

In the video game development industry, tasks related to design and specification require support to translate game features into implementations. These support systems must clearly define the elements, functionalities, and interactions of the game elements, and they must also be established independently of the target platform for its development. Based on a study for the specification of games that allows the generation of games as multi-agent systems, this work tries to check if the results can be cross-platform applied. As a case study the classic game Tetris has been used, a game whose very nature suggests that its implementation should be composed of vector and matrix data structures. The purpose is to validate the usage of a game specification based on multi-agent systems for the game's implementation on different platforms.

Keywords

Game development, Arcade games, Tetris, Multi-agent systems

6.1 Introduction

The design and specification of video game projects is a creative process that is often performed by people with no programming knowledge. These processes aim to translate design concepts into requirements and task definition. However, there is a lack of consensus on how to establish this process [AEMC08, AS10]. One of the first problems that come up are the constraints involved in designing for one platform or another depending on the required characteristics [AMW+13, BMR07]. In the literature, there has been a search for a framework to define the characteristics and functionalities of a game in an indirect way. For example, the field of artificial intelligence (AI) research in games has contributed advances with General Game Playing (GGP) [GLP05, PLST+16, Thi10, Thi11] where it describes in a way and manner that the same system is able to learn to play any game based just on the descriptions of its Game Description Language (GDL) [LHH+08, ELL+13]. However, these methods define specification systems that require high-level technical knowledge.

An alternative approach is to consider the game elements and their behaviors as autonomous entities that solve tasks assigned to them to ensure the correct execution of a game, similar to what would be done in the real world. In other words, by presenting an analogy between the elements of a game and the autonomous agents that constitute multi-agent systems (MAS) [DKJ18]. In Marín-Lora et al. [MLCSG20], a game engine able to generate games as MAS is presented. Where the game elements or agents have a set of properties and behavioral rules that allow them to interact with each other and with the social space they share, and where the definition of these behavioral rules is done by means of a formal semantics based on predicate logic. However, this work focuses on its own implementation and does not extrapolate its specification for other engines and other systems. Based on this game engine and its model, this study aims to validate the hypothesis that a game can be defined and specified as a system of interacting agents and that it is possible to implement it on different platforms. To this end, this work focuses on validating whether this model allows to define, specify and prototype a video game for multiple platforms in a fast and simple way. In addition, it is studied if it

is able to define and specify the behaviors of the game elements by establishing their logics. In an intermediate way between more traditional and artistic methods such as Game Design Documents (GDD) and other more technical and advanced methods such as GDL [ST09]. For this purpose, the study of this work goes through the implementation of a game on three different platforms: NetLogo, GDevelop and Unity [Wil99, GDe22, Uni22]. A MAS prototyping system, a 2D event-driven game engine and probably the most widely used game engine today, respectively. As a reference game, it is going to be used a game with a matrix nature and whose implementation a priori would not be conceived without the presence of the data structure of a matrix: the Tetris. The purpose of this case study is to validate the usage of a multi-agent specification for the implementation of games in different platforms.

The paper is organized as follows: Section 6.2 presents the state of the art studied for this article. Then, in Section 6.3, the data and game specification model will be presented. Subsequently, this model will be applied on the game of study in Section 6.4, and implemented on the three platforms in section 6.5. Finally, in Section 6.6, the conclusions obtained from the realization of this work will be presented.

6.2 Background

As in any other software design process, in video games there are multiple paradigms or design patterns to define the code structures that manage it and to establish the logic of the behaviors of its elements [Nys14]. Many of them are used to organize the assignment of responsibilities between elements or to define the behaviors and interactions between game objects. Some paradigms encapsulate the information needed to execute a method, perform an action or trigger an event; others are used in systems with one-to-many relationships between objects where, for example, if an object is modified, its dependent objects are automatically notified; or others that allow an object to change its behavior when its internal state is modified in a manner analogous to a state machine. Special mention should be made of iteration patterns that manage information flows. The commonest in video games is the game loop, that is, the continuous execution of the game that in each iteration processes the user interaction, the behaviors of the game elements and renders the scene in a continuous loop as long as the game state so indicates. A variant of this structure uses an auxiliary buffer as a storage method for the altered

information after each iteration in order to update it in the data model at the end of the cycle and thus keep the game information intact between the beginning and the end of the iteration. And to execute logical actions, the update model is also often used, based on an update function per game element, where each element evaluates in each frame its function at local scale. It is at this point where with the evaluation of the state of the game and the autonomous execution of actions of its elements according to their internal state, the correspondence between these patterns and the MAS occurs. MASs are composed of sets of agents that interact and make decisions among themselves within a shared environment [Woo09]. Within the shared environment, each agent has sets of properties and behavioral rules that determine its interaction with others. These agents have functions based on metrics associated with decision theory and game theory that allow them to exhibit autonomous, deliberative, reactive, organizational, social, interactive, coordinating, cooperative and negotiating behaviors [SCT03], that have traditionally been used in autonomous robotic systems to solve real-time problems. The selection of MAS as a reference system for the specification of video games is based on the analogy between the autonomous behaviors of agents and the elements that compose games. In other words, the behaviors and interactions in these systems have correspondences with the behaviors and interactions between individuals and their environment. MAS have aspects of formal specification to define a video game in a generic way, integrating specific aspects of the game such as the game logic with its entities and their behavior rules, or the game physics with the detection and response of collisions between game elements. However, it is obvious that the relationship between video games and MAS is not new. Multiple examples relating these two categories can be found in the literature: from the construction of elements for games, the interactions between their elements or their communication and cooperation protocols [Woo09, Pos07, MLCS19]. Also for more specific purposes such as the study of role-playing game (RPG) games [BBA+01], or to define games in which a large number of people participate in areas with different influences [ATE+12]. Currently, MAS and machine learning are already incorporated in several game engines, so they are also accessible to the general public [JBT+18, CMLRR20]. For this work, the focus has been placed on the application of this paradigm on game development, and specifically on the specification of its mechanics defined by means of scripts. Scripts are routines written in a programming language that provide an abstraction layer over the systems that manage the games in the dif-

ferent devices, that allow modifying the state of the games without the need of recompilation, and that are usually used for the management of the behaviors and for the management of the system events [And11]. Specifically, in video games, they are oriented to facilitate programming without actively thinking about optimizing the real-time execution of the game. During the last decade, the trend is towards the use of generic scripting languages, displacing languages specific to game development systems. Currently, the most widely used are C#, Python and JavaScript, and visual scripting systems such as Scratch or Unreal Blueprints [RMLRC18].

6.3 Video Games and Specification as MAS

The goal of this work is to test if the specification of a game based on the analogy between MAS and video games is able to be implemented on different platforms. For this purpose, it is necessary to establish a formal analogy between the concepts of a game and their corresponding concepts in a MAS. In addition, the method for the definition of a game must consider the features of the game, the elements that compose it, the definition of the behaviors and the user interaction. This work uses the game specification used by Marín-Lora et al. [MLCSG20] for their game engine. It is based on the definition of agent proposed by M. Wooldridge [Woo09], where an agent is a computer system located in an environment and capable of performing tasks autonomously to meet its design goals. By way of summary, some of the characteristics of the theoretical framework used for this proposal are detailed below:

- The environment to which the agents belong can be in any of the discrete states of a finite set of states E = [$e_0, e_1, e_2, \cdots$].
- The environment shared by all agents has a set of properties that determine its state and can be accessed by any agent in the environment.
- The agents have generic properties (geometric, visual, physical, etc.) and they also admit new properties to perform specific tasks.
- Agents have behavioral rules for modifying the state of the environment or the state of an agent in order to meet their plans and objectives.
- Agents have a set of possible actions with the ability to transform their environment Ac = [$\alpha_0, \alpha_1, \alpha_2, \cdots$].

- The run r of an agent on its environment is the interleaved sequence of actions and environment states $r : e_0 \rightarrow^{\alpha_0} e_1 \rightarrow^{\alpha_1} e_2 \rightarrow^{\alpha_2} \cdots e_{u-1} \rightarrow^{\alpha_{u-1}} e_u$.
- The set of all possible executions is R, where R^{AC} represents the subset of R that ends with an action, and R^E represents the subset of R that ends with a state of the environment. The R members are represented as R = $[r_0, r_1, \cdots]$.
- The state transformation function introduces the effect of an agent's actions on an environment τ: $R^{Ac} \rightarrow \wp(E)$ [FHMV04].

In order to transfer these concepts to a video game and to any support, it is necessary to define the general characteristics of the game and those of its elements such as those of an environment and its agents, respectively. Considering that there must be analogous functions and attributes for each element, regardless of the limitations or features of the game engine or software environment selected for its implementation.

Following the reference model, the rule specification is structured using first-order logical semantics [BLR92] based on two predicates: an IF condition and a DO action. Where each predicate executes calls to actions α of the system or evaluates arithmetic and Boolean expressions. The predicates specify the logic of the game so that the tasks to be performed by an agent are organized in predicate formulas where their elements can have the following predicate structures:

- **Action structure**: Composed of an atomic element including a single predicate literal DO.
- **Conditional structure**: Generated from the structure of the IF-THEN-ELSE rules [Kar88].

$$(IF \rightarrow \Psi) \wedge (\neg IF \rightarrow \theta)$$

where IF is a conditional literal predicate, and where Ψ and θ are sequences of new predicates that will be evaluated if the condition is met or if it is not met, respectively. The conditional predicate represents the evaluation element of a condition in the decision making process. Where the evaluation of the condition is based on the result of a logical expression that values the relationship between system entities. This logical expression may contain arithmetic expressions composed from system properties, game or agent properties, mathematical functions and numerical values.

$$IF(expression)$$

Based on the evaluation of these expressions, the logical elements determine the need for a game agent to perform an action α in the game. An α-action is defined as a behavior to be performed by an agent, and are formalized as non-logical function elements that can handle parameters such as arithmetic expressions. The set of actions is based on the create, read, update, and delete (CRUD) operations of information persistence [Dai10] applied to the game properties and its agents.

- **Create**: Creates a new agent, as a copy of an existing agent.
- **Read**: Reads the information of a game or a property of a game object. The syntax agent.property is used to read this information.
- **Update**: Modifies the value of a property of a set or an agent. The new value is determined from the evaluation of an arithmetic expression.
- **Delete**: Removes an agent from the game.

An example of these rules and the game specification could be presented as follows:

- AG1:
  - **Properties**: { *A: true, B: 1.00, C: "Hi!"* }
  - **Scripts**: { *IF(A)* $\rightarrow$ *DO(B = B + 1)* $\wedge$ *DO(C = "My name is AG1")* }
- AG2:
  - **Properties**: { *A: false, B: -1.00, C: "What is your name?"* }
  - **Scripts**: { *IF(B* $\leq$ *0)* $\rightarrow$ *DO(C = AG1.C)* $\wedge$ $\neg$ *IF(A)* $\rightarrow$ *DO(delete AG1)* }

From this model, the specification system designed must be able to define the behaviors of the elements that make up the sets in a general way.

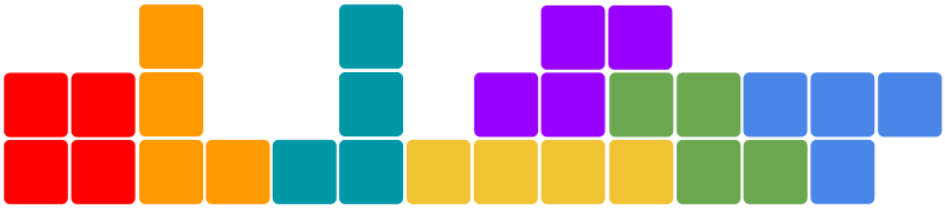

Figure 6.1: Diagram of game piece shapes.

6.4 Case Study: Tetris

The game to be used as a case study is Tetris. This classic logic arcade game was originally designed and implemented by Aleksei Pazhitnov in the Soviet Union and published in 1986 [Wil01]. The game consists of falling pieces composed of four blocks in different configurations. The goal of the game is to stack the pieces at screen bottom so that the accumulation of pieces does not reach the screen top. To avoid this, the player must position and rotate the pieces as they fall so as to complete as many horizontal rows as possible. Each time a row is completed, it vanishes and the blocks above it fall. Each time a piece is stacked, a random new piece appears from the screen top. There are seven variations of pieces in the game, each with a specific shape and named O, L, J, T, I, S, Z and T. Figure 6.1 shows a representation of them in the same order in which they have been described from left to right.

The data model of this game starts from the pieces, composed of four blocks arranged in a predetermined configuration. In addition, the player can perform geometric transformations on them to change their position and orientation. Finally, the game must eliminate blocks when they complete a row. Therefore, the types of agents needed for this implementation are three: the piece, the block and the checker. A representation of these three agents can be seen in Figure 6.2.

Piece Agent

Composed of four block agents in a prearranged layout. There is only one piece in the game at a time: the falling piece. While falling, the player can modify its position in a unit left, right and down using the corresponding arrow keys. In addition, he/she can rotate its orientation to the left and right with the L and R keys, respectively. As soon as it comes to rest with blocks already placed or with the background, it is

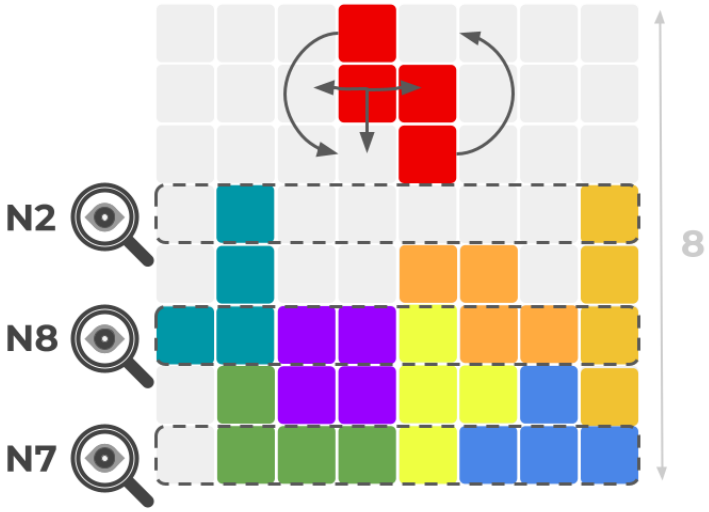

Figure 6.2: Diagram of game mechanics.

removed but the blocks that compose it are kept. These blocks will remain in their position until their line is completed or until the end of the game.

- **Properties**: { *resting: false* }
- **Movement script**: { $\neg$ *IF(resting)* $\rightarrow$ *(DO(y = y + 1)* $\wedge$ *IF(Game.KeyArrowLeft)* $\rightarrow$ *DO (x = x - 1)* $\wedge$ *IF(Game.KeyArrowRight)* $\rightarrow$ *DO (x = x + 1)* $\wedge$ *IF(Game.KeyArrowDown)* $\rightarrow$ *DO (y = y + 1)* $\wedge$ *IF(Game.KeyR)* $\rightarrow$ *DO (angle = angle - 90)* $\wedge$ *IF(Game.KeyL)* $\rightarrow$ *DO(angle = angle + 90)* }
- **Rest script**: { *IF(collisionRest)* $\rightarrow$ *(DO(resting = true)* $\wedge$ *DO(delete))* }

Block Agent

They initially compose a piece and move with it. By default, they are considered to have a dimension of one unit. When the piece goes to rest state, they are unlinked from it and remain static in their waiting position. It has a property to store the information of the row for the moment in which it is at rest. If they come across a check agent in elimination mode, they must communicate to the piece above them that it has to move down one position, and then it must be eliminated from the game.

- **Properties**: { *resting: false, row = -1* }
- **Set script**: { *IF(resting)* $\rightarrow$ *DO(row = y)* }

- **Destroy script**: { *IF(collisionCheck)* $\rightarrow$ *IF(check.deleteMode)* $\rightarrow$ *((IF(collisionBlockUp)* $\rightarrow$ *IF(blockUp.y > y)* $\rightarrow$ *DO(blockUp.y = blockUp.y + 1)* $\wedge$ *DO(delete)) }

Check Agent

In each of the rows there is a controller agent that checks the number of blocks occupying its row. When activated, it runs from left to right through its row checking if each position is occupied by a block. If when it reaches the end of the row its number is equal to the number of existing columns, it must return in the opposite direction informing all the blocks in its row that they must be destroyed, and it must send a message to all the checker agents in the rows above it to move the blocks at rest one position down.

- **Properties**: { *count: 0, deleteMode: false* }
- **Forward script**: { *IF(x* $\leq$ *Game.width)* $\rightarrow$ *DO(x = x + 1)* $\wedge$ $\neg$ *IF(x* $\leq$ *Game.width)* $\rightarrow$ *DO(x = 0)* }
- **Backward script**: { *IF(x > 0)* $\rightarrow$ *IF(deleteMode)* $\rightarrow$ *DO(x = x - 1)* }
- **Check script**: { *IF(collisionBlock)* $\rightarrow$ *DO(count = count + 1)* }
- **Clear script**: { *IF(count == Game.width)* $\rightarrow$ *DO(deleteMode = true)* }

6.5 Results and Discussion

From this specification, a version of the game has been successfully implemented on the three intended systems: NetLogo, GDevelop and Unity. All three have respected the game description and the game definition as MAS. The most notable particularities of these implementations reside mainly in the graphical interface available and in some features in the way of composing the game logic. In the case of NetLogo, and in contrast to the other two systems, there is no graphical editor to create the initial spatial layout of the game interactively and it has been necessary to create this initial configuration through its API and its scripting system based on the turtle language. Moreover, since this system is not intended to generate games, its graphic quality has been reduced to regions of colored pixels. This system has the particularity of having "broadcast" functions for one type of agent, so that communication could occur in a more direct way. However, the model has been followed and broadcast through the use of auxiliary variables.

At the logical level, the major particularity has been the rotation of the pieces. It has been chosen to follow the methodology used by an example of the game in its resource library, where the rotation occurs through the exchange of positions between pieces from the definition of a central block agent defined as block 0. In the case of GDevelop, the game has been arranged in its web editor and the logic has been implemented through its logic system based on event management. In contrast to the previous case, the graphical level of the system has allowed to generate more visual forms. The most outstanding particularity of this system in terms of logic has been the communication between agents. For example, after collision events it has been necessary to create auxiliary variables in the general properties of the game and to subscribe potentially interested agents to these variables. Finally, Unity is the most powerful of the three environments. It has made it possible to compose the game and its specification from its editor and its scripting system in the C# language. The particularities of this implementation are very similar to those found in GDevelop, where communications have been made through game variables and each of the agents checked its status after each iteration of its Update function.

6.6 Conclusions

The work presented in this paper aims to validate the hypothesis that a game can be defined and specified as a system of interacting agents and that it is possible to implement it on different platforms. For this purpose, a model for the specification of games based on a game engine created to generate games as multi-agent systems has been taken as a starting point. From which it has been studied and tested whether the specification of a game with this model can be implemented on different platforms. For its validation, the classic game Tetris, a game that by nature should be based on vector and matrix structures, has been used as a reference. Finally, the game has been defined and specified according to the reference model obtaining a total of three different agent types for the game composition. With this specification, the same system has been implemented in three different platforms with satisfactory results. With this, it can be said that the starting hypothesis has been successfully validated and the objectives of this work have been met. As a future work, the extension of the specification system is being considered through the definition of a formal language that would allow the specification and programming of games following this same model based on MAS and based on first-order logic.

Part IV

Serious Games Development

Chapter 7

Improved Perception of Ceramic Molds Through Augmented Reality

Publication

Marín-Lora, Carlos., Sotoca, Jose. M., & Chover, Miguel. (2022, Apr). Improved Perception of Ceramic Molds Through Augmented Reality. Multimedia Tools and Applications. (Q2).
https://doi.org/10.1007/s11042-022-13168-5.

Abstract

Augmented Reality techniques allow the user to visualize part of the real world through a display device by incorporating graphical information into the existing physical information. In this sense, it is important to know how the physical presence of the user in the augmented reality experience can affect the perception and evaluation of the product. To this end, this work presents a theoretical framework that explains how users perceive and evaluate the benefits and quality of augmentation with augmented reality through their physical presence, compared to visualizing the same experience through a video. The application was developed for the exhibition and sale of ceramic molds. Users viewed graphical information about the mold, placed between them and the screen while seeing themselves in the television as if it was a mirror. The experiments showed that the integration of the product into the environment and the spatial presence of the users had a positive effect on the

perceived value in terms of usefulness and enjoyment, improved comfort in the purchase decision, and reinforced the overall opinion of the product.

Keywords

Augmented reality, product demonstration, spatial presence, decision comfort, ceramic molds

7.1 Introduction

Augmented reality (AR) is a visualization technology that allows enriching real-world images by inserting synthetic elements that are integrated into them [Azu97]. Since the first research on the field back in the 1960s [Sut65] to its extension to the general public in the 2010s [BCL15], this technology has been applied in fields as diverse as education [Lee12], medicine [ZHMZ14] and engineering [dSCMZ20], among others.

AR has been and is used to improve the representation of content, either through its static arrangement in the world or dynamically with animations. One of the fields where its potential has been most exploited is product sales and promotion, where it has been detected to have a positive effect on decision comfort, purchase satisfaction, and brand positioning [BT10]. This not only has a short-term effect on a specific product but by its contribution to consumer satisfaction, purchase loyalty upgrade, buyer retention and word-of-mouth effect [HdRC+17].

This work presents a study of a user experience with an AR audiovisual application for the exhibition and promotion of molds in the ceramic industry. In this AR experience, an avatar shows up explaining the most outstanding elements of the mold through an animated scene and with sound effects, while performing the procedures for which they have been designed. The purpose of the application is focused on visually demonstrating how the mold technology works and attracting the attention of the visitors. This is done through the virtual arrangement of a scripted sequence on the projection of the mold.

From an analytical standpoint, the goal of this work is focused on technically determining what benefits, and to what extent an AR experience helps to improve product awareness and marketing. In addition, it is analyzed to what extent the spatial presence of users during the AR experience influences a more positive

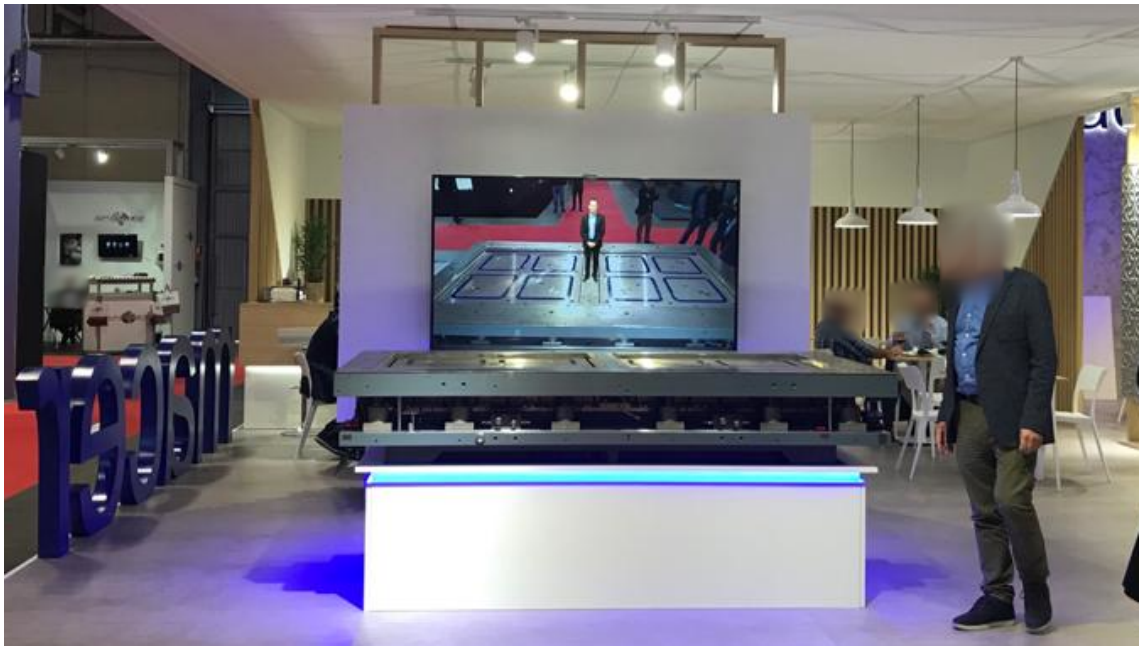

Figure 7.1: Application and ceramic mold at the advertising company's stand.

evaluation of the product, along with the utility and hedonic benefits, increasing the quality feeling about the product and the decision capacity.

The paper presents a comparison of feedback from users who participated in the AR experience versus those who only viewed the experience through a video. All this is analyzed with a Structural Equation Modeling (SEM) that takes into account different aspects that influence users' expectations. This model straightforwardly explains user behavior, although there are alternative models with more variables and feature clustering techniques that could be employed [LCSP12, RFBS20, RBF21, RBNF21, BBS19].

The context for which the application was designed consists of a ceramic mold placed in the center of a trade fair stand next to a large television monitor and a high-resolution camera. Figure 7.1 shows an image of the application context inside the stand of the company, which was located at the International Fair for Ceramics (CEVISAMA) [Cev20].

As a summary, this paper is organized as follows. Section 7.2 presents a brief state of the art on AR within the field of engineering and marketing, which is the associated context to this paper. In Section 7.3, the hypotheses and general objectives of the study carried out on the experience are presented. Section 7.4 describes the ceramic

mold, the main aspects of the application, and the elements that have contributed to its design and development. Section 7.5 shows the different parts that were developed to implement the AR application. The protocol used to evaluate the effectiveness of the application by performing user tests and studying their results is described in Section 7.6. A brief discussion of the implementation and its impact on users is included in Section 7.7. Finally, conclusions and possible future lines are presented in Section 7.8.

7.2 State of the Art

A widely accepted definition of AR is given by Azuma [Azu97] as a system that enables the user to see the real world, with virtual objects overlapped or composited. To this end, the author defines three aspects: the combination of real and synthetic elements, real-time interaction, and the alignment of all elements in a 3D world. Accordingly, AR complements reality rather than replacing it so that virtual and real objects coexist in the same space. In this sense, Carmigniani et al. [CFA+11] emphasizes that AR does not create an artificial reality that replaces the real environment, but rather overlaps additional information on real environments or objects.

Currently, AR technology is under development and there are still challenges to be solved in different aspects of its implementation and applications [VKP10]. Simultaneously, AR plays an important role in visualizing information about training and manufacturing processes [DVRL12], education [CLCH17, AA17], cultural heritage [AC19] and product presentation [LS07].

From an engineering and industrial processes standpoint, technologies such as AR are applied in multiple fields [LNO17]. Their applications range from learning how to use industrial tools, the analysis of protocols within factories, to process training simulators [Pae14, CKW13]. Additionally, there are concrete examples such as a training tool for the maintenance of high voltage lines [GBF+16], an industrial assistant within the processes of a shipyard [FLFCBNVM18], a tool for use in the automotive field [JER18], a solution for the inspection and analysis of architectural elements [WFM+96], and a marketing element in the sales process of products [COEGD+19].

Within the field of marketing and product visualization, it is easy to find multiple uses: as an advanced product visualization tool in a physical store via smartphone [MP17], as a method of previewing the arrangement of furniture [GYSE20, RFH19],

or to establish settings in the presentation of products during their online retailing [Dac17]. However, at present, AR is not yet a widespread element in physical store shopping processes, and its effectiveness on the shopping process is still under evaluation [YCS17, CCE+10]. Some studies try to validate whether their inclusion improves brand reputation and user experience, and thus increases consumer conversion rates [RFH19]. By adding these functionalities to the user experience, retailers expose themselves to a paradigm shift in the way their potential customers interact with their products and with the decisions they have to make before and after agreeing to purchase the product [SK14, HdRC+17].

7.3 Hypotheses

At a practical level, the incorporation of AR seeks to add value to the product to be marketed, trying to highlight its characteristics and provide a visual concept in its promotion and sales process, something new in the ceramic machinery industry. However, despite their inherent relationship, two aspects can also be distinguished in the objectives of this work. The first one is focused on a commercial and promotional perspective of the company, and the second one is focused on a technical analysis of the performance provided on the potential consumers of a product. This paper focuses on the analysis and discussion of the second one, following previous works done about user experiences and customers' interaction with products, and decision-making processes.

Thus, inspired by the proposal of Hilken et al. [HdRC+17], this model raises four factors that determine the attitude of users towards product quality: *Hedonic*, *Usefulness*, *Spatial Presence* and *Purchase Intentions*. In addition, two latent variables are included in the model that act as intermediaries: the *Perceived Value of Experience* and the *Product Decision-Making*, that act on the final opinion of the product (see figures 7.6 and 7.7).

7.3.1 Aspects Related to the Perceived Value of Experience

The study is based on feedback from participants of the experience created for the trade fair where the mold was exhibited. After experiencing the AR application, data was collected to estimate their degree of satisfaction with the product. This information was compared with another group of independent users who experienced

the same sequence about the ceramic mold through a video.

In this sense, the first level of experiential knowledge about service products arises in terms of *Usefulness* and *Hedonic* values [BFH06, BLKG05], where a first impression about the functionality and performance of the product and the experiential enjoyment provided by the service experience is collected. Accordingly, in the work developed by Childers et al. [CCPC01] it is shown that users value utility and enjoyment when purchasing a product. Recent works following this approach suggest that the use of AR improves the perception of these two terms when purchasing a product [PM14]. All this is supported by the impression generated in the user about the functionality and performance of the product and the experiential enjoyment provided. In this sense, the following hypotheses can be formulated:

H1. *Hedonic* enjoyment has a positive reaction on the perceived value of the AR experience.

H2. *Usefulness* has a positive reaction on the perceived value of the AR experience.

Besides that, AR makes it possible to generate applications with environmental embedding, in the sense of displaying virtual content in the person's real environment and adding a control through physical simulation, so that users can interact with the synthetic elements of the AR experience. In this work, the user only acts passively observing the features of the product that is explained through an avatar, so in the experience, there is only environmental integration of the users who observe the virtual content that shows up over the ceramic mold.

Another aspect to consider is the correspondence between the synthetic elements and the real world through *Spatial Presence*. It implies that people perceive and experience inputs as if they were real even though some of them are not. If this sense of *Spatial Presence* is achieved, users feel located in the real world, even though the virtual elements displayed are no more than an illusion [HWV+15]. In the work by Klein et al., [Kle03] it is shown that the feeling of *Spatial Presence* increases customers' beliefs and attitudes about product attributes. Focusing the analysis on the sense of *Spatial Presence* perceived in a real environment and the effects that this experience causes over the visitors, this third hypothesis is proposed:

H3. The *Spatial Presence* of users in the display of synthetic elements increases the perceived value of the AR experience.

In this work, an avatar carries out an understanding of the ceramic mold features while visually showing, through synthetic elements, the different parts of the mold. In this sense, a comparison could have been made about whether the visual information was more relevant than the verbal one. This could affect users in their way of processing the information. However, it was decided not to incorporate this aspect in the work since the language shown was quite technical and some users did not have this level of expertise.

7.3.2 Aspects Related to the Product Decision-Making

One aspect to validate is the degree of satisfaction reported by the customer after deciding to purchase the product. Parker et al. [PLX16] define this concept as "decision comfort" or the degree to which the consumer feels pleased with the decision made. Decision comfort constitutes a soft-positive affective response that explains customers' changes of opinion in their decision making, beyond generic affect and decision confidence. The latter reflects the level of certainty about making the best decision, based on a cognitive evaluation of the pros and cons around the decision. As the same authors state [PLX16], decision comfort is an affect-based sense of ease related to the process of making the decision.

These possible changes in decision-making are influenced by the way of presentation and the initial satisfaction of the user with the product and ultimately influence the *Purchase Intentions*. For example, Li et al. [LDB+05] showed that a 3D presentation of products significantly and positively impacts online *Purchase Intentions*. In other works, the authors apply AR techniques to enrich user experience when shopping for a product in a physical store [COEGD+19].

Furthermore, AR enhances the feeling of *Spatial Presence* in an environment and the virtual objects rendered in the experience [Sch09]. It is felt that in the case of AR on a ceramic mold, the fact that users can see themselves on the screen influences the processes of unconscious spatial cognition, making presented virtual objects look real and it can affect the *Product Decision-Making*. This allows the following two new hypotheses to be established:

H4. The *Spatial Presence* of the user has a positive effect in the *Product Decision-Making*.

H5. The *Purchase Intentions* have a positive effect in the *Product Decision-Making*.

7.3.3 Result of Combining Perceived Value of Experience and Product Decision-Making

In the proposed model, it is given by the combined action between the perceived value of the AR experience and the *Product Decision-Making* process, which determines its final satisfaction. Both concepts are defined by latent variables that try to collect different factors taken into account by users and whose combination is compared to the last question of the questionnaire that collects user satisfaction about ceramic mold. With all this, the context in which the experience was developed is presented below. Furthermore, an attempt will be made to check the validity of the hypotheses raised in this section.

7.4 Description of the Exposed Product

A ceramic mold is an industrial engineering component that allows shaping tiles during the pressing stage. The appearance and quality of the finished product depend directly on the functioning of the mold. For its design, the features of the ceramic pieces to be produced (material, dimensions, relief, and decorations) and the pressing operations to be performed on it must be taken into account. Ceramic molds can be classified into five types according to their structural and functional characteristics: penetrating molds, penetrating mirror molds, MSF/SFS double mirror molds, quick-change molds (CRS), and 380/440V molds. Although there are multiple types of molds, most of them share components although there are important differences between them, both in structure and operation [Gal08].

The product on display at the fair and described in this paper is a MSF/SFS double mirror mold, which features a penetrating-type top arrangement. This double die system allows the formation of tiles with a spacer and with the thin side of the spacer facing up. The upper part is used with presses of 10,000 tons of pressure through an incorporated isostatic system and supported by a guiding system working along with a compensation system that allows maintaining a balanced pressure between the different cavities. Additionally, this mold incorporates state-of-the-art elements that improve its reliability in the event of accidental maneuvers: quick electrical connections, internal cleaning system, anti-wear plate, and isostatic punch holder. The advantage of the MSF/SFS double mirror mold is its application to any type of special molding and in any technology.

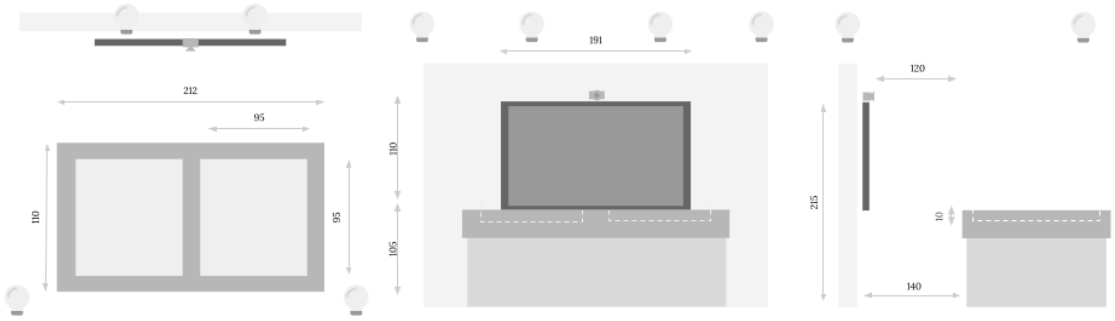

Figure 7.2: Dihedral representation of the application environment.

The physical environment of the presentation is in the context of a commercial stand of approximately 20 square meters hosted at CEVISAMA [Cev20]. At the center of the location is the ceramic mold measuring 212 by 110 centimeters, resting on a stand 105 centimeters high. Visitors can approach and interact with the mold at will. About 120 centimeters inward from the location, and placed on a wooden panel, rests a high-resolution television covering a space of 191 by 110 centimeters, with its top 215 centimeters above the floor. At this upper point is located a high-resolution camera oriented towards the mold and the outside of the stand, giving a perspective of the scene that produces a mirror effect on visitors. Figure 7.2 shows a diagram in dihedral perspective of the layout of the physical environment of the experience.

The description of the features of the ceramic mold is based on a script provided by the advertising company where the technical characteristics of its ceramic mold are exposed. The storyline of the presentation is led by an avatar that presents each element of the mold in its augmented reality representation of the real mold. As a whole, the application consists of a 10-minute cyclic sequence.

The mold elements that are highlighted are:

- **Removable blades**: Elements integrated into the edges of the raw material containers. They are in charge of preserving the integrity of the edges of the container and the ceramic pieces.
- **Anti-wear plate**: Device located in the contact areas between the upper and lower elements of the mold, to protect the components during pressing operations.
- **Internal cleaning system**: System based on air pulses to eliminate the rest of

the material retained in the mold after each operation.

- **Quick electrical connectors**: Mold electrical outlet connection rack to provide safety and failure protection.
- **Sliding system**: Device in charge of extracting the ceramic piece from the mold with a system of guides and pistons that lift the part out of the punch and the mold.
- **Isostatic system**: System that manages the force compensation applied by the press on the mold to operate more uniformly on the final part.

7.5 Implementation of the AR Application

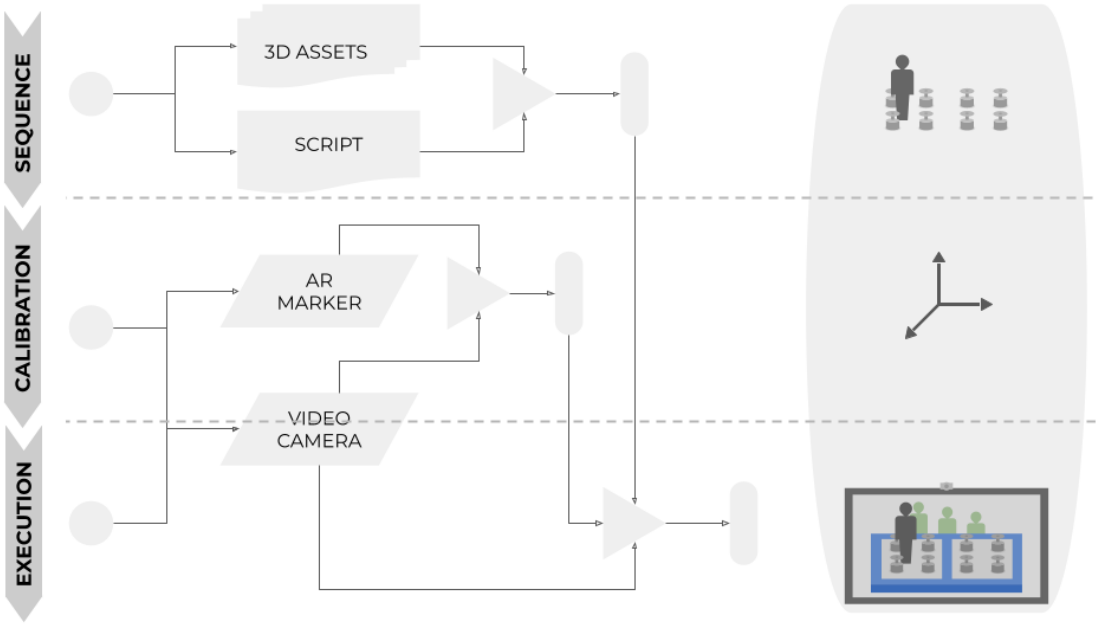

Figure 7.3: Flowchart describing the composition and integration of the three phases that compose the design and real-time execution of the AR application.

The application's purpose is to transform a script about the commercial demonstration of the operation and characteristics of a ceramic mold into an AR system. For this purpose, three phases are defined within the development of the application: the transformation of the script into a 3D sequence, the calibration of the AR

system, and its integration in a single execution. Figure 7.3 shows a diagram with the elements that compose these three phases.

At the technical level, the application generated for this experience was developed on the Unity game engine [Uni21] along with the OpenCV computer vision API in its plugin version for Unity based on markers (OpenCV for Unity v2.3.7). And at the hardware level, the kit available for the implementation and execution of the application consists of a mid-high-end computer (HP Z420 - CPU Intel Xeon E5-1660, 3.70GHz, 32GB, NVIDIA Quadro K5000), a high-resolution USB camera (Logitech Brio 4K Stream Edition) and a high-resolution television (Sony KD85XG8596 85" Ultra HD 4K Android TV).

7.5.1 Scripted sequence and synthetic elements

The script developed by the marketing company is composed of a cyclical sequence of stages to promote key features of the product, all linked by the presence of a narrator. Each of the stages of the sequence is composed of 3D elements generated from different sources.

The narrator is integrated into the action in video format. The production of the videos took place in a recording studio with an actor placed on a chroma key. The post-processing of the chroma key to remove the green background is done using a shader implemented for this application.

The assets used in the application to represent the mold components were modeled based on the manufacturer's technical specifications using 3DS Max. Similarly, the animations designed for their integration into the sequence were planned to take into account the actual arrangement of the elements in the mold, their operation, and range of motion, trying to generate fluid and visually appealing movements. The materials used for the synthetic elements of the mold were parameterized so that they resemble as closely as possible the real elements of the mold in the lighting conditions of the scene at the fair. That is, the materials with metallic properties and in dark tones try to simulate the visual perception of elements such as steel or tungsten that are part of the real mold materials. As examples, Figure 7.4, the 3D models of three parts of the mold are arranged from left to right: the sliding system, the quick electrical connectors, and the anti-wear plate.

To technically implement the script provided by the advertising company, a sequence of actions was generated within a Unity scene. These actions are controlled

Figure 7.4: Examples of some of the 3D models representing the mold components.

sub-sequences for the input and output of the elements that are intended to be highlighted in the script. Each sub-sequence is stored inside a Unity game object, with its elements and animations, and is managed by a general sequence controller that activates or deactivates it accordingly.

As can be seen in the first phase of Figure 7.3, the outcome of this phase is a 3D sequence generated in Unity.

7.5.2 Vision Module and System Calibration

A key element in any AR system is the correspondence between the real and synthetic worlds. In the literature, multiple libraries provide the necessary tools to establish this correspondence, such as ARCore [Goo21], Vuforia [Vuf21], and OpenCV [BK08].For this experiment, one aspect to take into consideration is that the relative position between the camera and the mold remains constant once the system is calibrated. Since the camera is anchored to the television set placed on wall support, the mold is a static element of large tonnage.

Given these conditions, and taking into account the dimensions and geometry of the mold and the particularities of the experience, it is impossible to have a continuous mark that determines the reference system for positioning the virtual objects. However, given the static situation of the system, it is possible to use a previous calibration to obtain the reference system that allows establishing a correspondence between the 3D points of the physical mold plane in different positions and its projection in the image integrating the synthetic world and the real world. This information, captured by the real-world camera, is captured in a plane on which the video texture is applied from the USB camera input.

For this process, the ArUco module implemented in OpenCV was used to calibrate the camera by obtaining the intrinsic and extrinsic parameters as well as their distortion coefficients. For this purpose, a correspondence is established between

Figure 7.5: Example of integration of elements in the AR application and with the environment and visitors to the stand.

the points belonging to the environment and their projection on the camera image. With this module, calibration can be performed from the corners of a marker or the corners of the ChArUco [GJMSMCMJ14]. This calibration method is more versatile than traditional methods based on checkerboard patterns and allows for occlusions or partial views. In this way, it is possible to detect the mark, to know its position and rotation, and to be able to project on the mark 3D objects accurately. In this way, it is possible to determine a reference plane on which to position the synthetic world.

7.5.3 Integration and Execution in Unity

Finally, with the sequence of actions generated and the reference system obtained, it only remains to integrate these elements in a final application that executes a continuous loop and integrates the elements in the real environment of the fair.

From the reference system, the projection is arranged between the video input coming from the camera and the synthetic elements that make up the sequence. As can be seen in the last phase of the diagram of Figure 7.3, the result intends the video with the sequence.

As a technical note, to achieve the effect of occlusion of the synthetic elements when they "emerge" from the mold, an occlusion shader was implemented from the synthetic geometries of the elements present in the scene. An example of this case can be seen in Figure 7.5, with the pistons that lift the ceramic pieces coming out of the mold.

7.6 Experimentation and Analysis of Results

7.6.1 Experiments Protocol

One of the goals of this work was to test if the use of AR improves the perception of product features. For this purpose, two experiments were carried out. The first experiment was planned to collect data to estimate the degree of satisfaction that the user perceived about the product after visualizing the AR experience. During the experience, participants had freedom of movement around the ceramic mold, being able to contemplate the physical model and visualize the synthetic elements integrated with the camera video image observed through the television screen. Additionally, a second experiment was arranged to measure the reality-enhancing effect of an AR experience and analyze how it affects *Spatial Presence*. This second experience was conducted with another group of users, different and independent from AR ones. For this case, participants watched the playback of the sequence through a video on a screen.

In both groups, users enjoyed the experience for 10 minutes. In the end, they were asked to fill out a questionnaire. The questionnaire consisted of 10 questions to express their impressions, rating each on a five-point Likert scale ("very unfavorable" = 1 to "very favorable" = 5).

The test was carried out on a population of 50 people for the AR experience and 50 people for the video experience. In the first group, 32 were men and 18 were women with ages ranging from 19 to 55 years old, among the visitors who came to the stand and voluntarily took the survey. In the second group, 34 were male and 16 were female, and respondents ranged from 20 to 51 years old.

The test questions were intended to test the validity of the experience and the hypotheses stated above. The questions were designed to express impressions of the AR experience at four levels: *Hedonic*, *Utility*, *Spatial Presence* and *Purchase Intentions*. The definition of the questions was based on the analysis of questions

on different aspects raised in the literature on users' perception of an AR experience [BC18a, CCPC01, HdRC+17, RFH19, VWG+04] (see Table 7.1). Moreover, a final question was included to measure the user's overall satisfaction with the product.

In Table 7.1, the description of the questions is shown in Column 1. Columns 2 and 3 show the mean score and standard deviation for users who watched the video experience (Video), and for users who watched the AR experience (AR). The grouping of the test questions into latent constructs is also shown.

Overall, the results presented in Table 7.1 reveal that users' ratings of the different questions were better for the AR experience than the experience through video. These differences are clearly seen at the *Hedonic* and *Spatial Presence* levels. In the case of the questions related to *Utility* and *Purchase Intentions*, these improvements are not as significant as users take into account other factors such as product features. In the case of final *Product Satisfaction*, there is also a clear improvement from Avg_{video} = 3.1 to Avg_{AR} = 4.4.

Questions	**Video**	**AR**
Hedonic adapted from [HdRC+17]		
Q1. I found the viewing experience entertaining.	3.0 ± 1.1	4.2 ± 0.7
Q2. The sequence has been funny.	2.8 ± 1.2	3.8 ± 1.1
Usefulness adapted from [CCPC01, HdRC+17, RFH19]		
Q3. The sequence provides information on its operation in a short time.	3.9 ± 0.8	4.4 ± 1.0
Q4. The sequence allows me to more clearly appreciate the characteristics of the mold.	3.7 ± 1.0	4.4 ± 0.7
Spatial Presence adapted from [VWG+04, HdRC+17]		
Q5. I felt that the experience could fit into the real world.	2.8 ± 1.2	4.1 ± 0.9
Q6. I have perceived that the elements of the mold were integrated in a realistic way.	3.1 ± 1.0	4.1 ± 0.7
Q7. I felt the action seemed real.	2.2 ± 0.9	4.1 ± 0.8
Purchase Intentions adapted from [BC18a, HdRC+17]		
Q8. If I had to buy the mold, the visual experience has helped me make a decision.	3.2 ± 1.1	3.9 ± 0.9
Q9. The characteristics of the product are interesting as a purchase option in the ceramic industry.	3.6 ± 0.7	4.1 ± 0.8
Product satisfaction		
Q10. My general opinion on the ceramic mold has improved.	3.1 ± 1.0	4.4 ± 0.8

Table 7.1: Questionnaire and overview of constructs. It is shown the average score and the standard deviation for users for video and AR experiences.

	1	2	3	4	5	6	7
1. Hedonic	1.00						
2. Usefulness	0.72	1.00					
3. Spatial Presence	0.43	0.45	1.00				
4. Purchase Intentions	0.48	0.50	0.77	1.00			
5. Perceived Value of Experience	0.83	0.86	0.52	0.58	1.00		
6. Product Decision-Making	0.48	0.50	0.77	1.00	0.58	1.00	
7. Q10	0.55	0.57	0.69	0.88	0.66	0.88	1.00

Table 7.2: Relation between independent latent variables KSI and dependent latent variables ETA for the video experience.

	1	2	3	4	5	6	7
1. Hedonic	1.00						
2. Usefulness	0.30	1.00					
3. Spatial Presence	0.66	0.20	1.00				
4. Purchase Intentions	0.73	0.22	0.53	1.00			
5. Perceived Value of Experience	1.00	0.30	0.66	0.73	1.00		
6. Product Decision-Making	0.89	0.27	0.65	0.81	0.89	1.00	
7. Q10	0.91	0.27	0.65	0.80	0.91	0.98	1.00

Table 7.3: Relation between independent latent variables KSI and dependent latent variables ETA for the AR experience.

7.6.2 Hypothesis Testing

To evaluate the hypotheses proposed in Section 3, a SEM was run with the program LISREL 10.20. A maximum likelihood model was applied to generate the model using robust estimation, considering that the data do not necessarily follow a normal distribution [JOW16]. Using this methodology, the different constructs represented by the independent latent variables KSI (*Hedonic*, *Usefulness*, *Spatial Presence* and *Purchase Intentions*) associated with the questions in Table 7.1, could be grouped into the dependent latent variables ETA (*Perceived Value of Experience* and *Product Decision-Making*). These dependent latent variables act as intermediaries collecting different constructs that are combined to establish a relationship with the final satisfaction of the product (question Q10). The grouping mechanism is determined by a model that obtains a better fit. The proposed model was inspired in [HdRC+17] and [RFH19] although the AR experience proposed in both articles are not the same as the present work. From these latent variables, Tables 7.2 and 7.3 show the

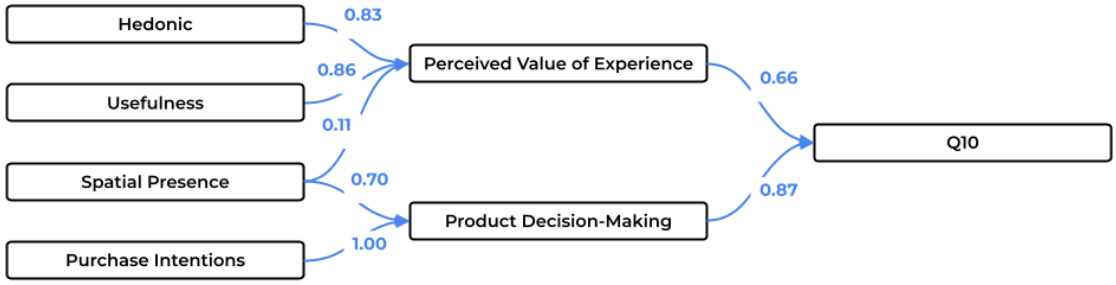

Figure 7.6: Model to explain the degree of satisfaction with the product through the video experience.

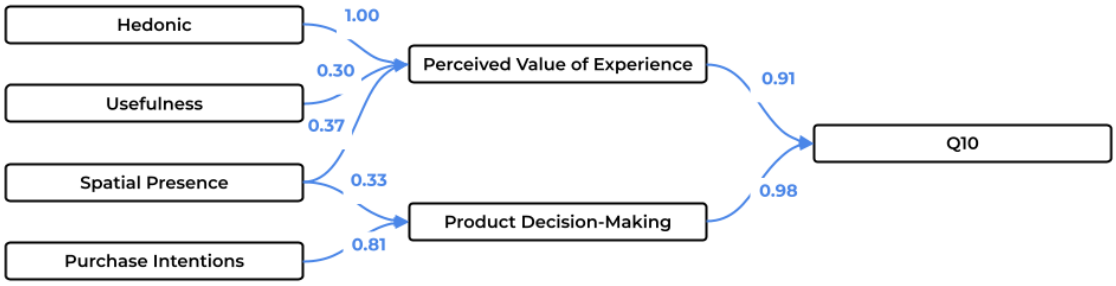

Figure 7.7: Model to explain the degree of satisfaction with the product in AR experience.

relationships between the different constructs on Q10. In these tables, it can be seen how for instance the *Hedonic* component affects the scores on Q10 in a different way in the AR experience (0.91) than in the video experience (0.55).

Figures 7.6 and 7.7 show the model for the case of the experience with video and AR, respectively. The same model was applied to the test data obtained from both groups of users, those who saw the features of the ceramic mold from a video and those who saw them in AR. The goal was to evaluate the differences that arise in the model based on the feedback received by both groups of users.

The results on the proposed model in Figures 7.6 and 7.7, show that the *Perceived Value of Experience* is influenced by both constructs *Hedonic* (β = 0.83; p $=$ 0.001) and *Usefulness* (β = 0.86; p $=$ 0.001) for the video experience, and *Hedonic* (β = 1.00; p $<$ 0.001) and *Usefulness* (β = 0.30; p $<$ 0.001) for the AR experience. In this sense, hypothesis H1 and H2 were validated. Additionally, by looking at the model in the AR experience, it can be seen that *Hedonic* expectations reinforce the *Perceived Value of Experience* more than viewing through video. However, the *Usefulness* has

	Video		AR	
	AVE	**CR**	**AVE**	**CR**
Hedonic	0.59	0.62	0.35	0.27
Usefulness	0.40	0.34	0.26	0.16
Spatial Presence	0.51	0.61	0.74	0.86
Purchase Intentions	0.35	0.27	0.58	0.61
Perceived Value of Experience	0.48	0.57	0.41	0.46
Product Decision-Making	0.75	0.81	0.38	0.32
Q10	0.61	0.65	0.89	0.94

Table 7.4: Average Variance Extracted (AVE); Composite Reliability (CR) for the case of video and AR experiences.

given higher values in video than AR for this construct.

Hypothesis H3 states that the *Spatial Presence* of users in the display of synthetic elements increases the *Perceived Value of Experience*. In the case of video, the model indicates a value of (β = 0.11; p $=$ 0.540) for the *Spatial Presence*. In this case the *Spatial Presence* was not significantly linked to *Perceived Value of Experience*, rejecting H3 for video. For the case of AR, the value obtained is (β = 0.37; p $<$ 0.001) complying with H3.

It should be noted that the *Spatial Presence* also intervenes in the *Product Decision-Making* (hypothesis H4). When observing the models in Figures 7.6 and 7.7, the data indicate values of (β = 0.70; p $<$ 0.001) and (β = 0.33; p $<$ 0.001) from video and AR respectively. In this case, both experiences verify hypothesis H4, obtaining a higher value of the *Spatial Presence* influence for the video viewing. It is possible that the lack of spatial presence in video has a negative effect on purchase intentions and that it would be related to the low values given to the questions for the case of video viewing in Table 7.1.

In the case of *Purchase Intentions*, this construct provides direct information in the *Product Decision-Making* process. In the case of video, this variable is decisive (β = 1.00; p $<$ 0.001) while in AR its value decreases (β = 0.81; p $<$ 0.001). In both experiments, it can be stated that hypothesis H5 is confirmed.

Finally, both in the case of video and in the case of AR, the dependent latent variables *Perceived Value of Experience* and *Product Decision-Making* have a similar importance in predicting product satisfaction (question Q10). This is a delicate aspect to assess and the importance of both variables could vary depending on the context in which the experience was carried out, although it seems that both

aspects significantly affect product satisfaction.

Table 7.4 shows the latent variables including the question Q10, that were used for the case of video and AR experiences. Also included in the Average Variance Extracted (AVE) and the Composite Reliability (CR) for each variable. This allows measuring the internal consistency between the constructs within the questionnaire and their relationship by generating dependent latent variables.

Table 7.5 shows the model fit results for the test data generated from users who saw the experience in video and AR, respectively. The reference values that indicate whether the model has achieved a good fit are shown in the last column of the table [MLV14]. In the case of the parameter χ^2/df (Chi-squared divided by the degrees of freedom), its value is relatively low in both experiences. This parameter is sensitive to sample size and the number of degrees of freedom of the model.

In the case of the descriptors Root Mean Square Error of Approximation (RMSEA) and Root Mean Square Residual (RMSR), the AR experience got a reasonable fit, RMSEA = 0.09 and RMSR = 0.08. For the parameters Goodness of Fit Index (GFI) and Adjusted Goodness of Fit Index (AGFI) parameters, in AR relatively high values are achieved, especially for the Goodness of Fit Index, GFI = 0.90 in the range of values for a good fit. This decreases for the case of AGFI that is adjusted by the degrees of freedom of the model.

For the group of users who viewed the experience via video, the data collected for the model through the user test fit similarly to the AR case. In this sense, it can be indicated that the relationships between the latent variables in the model for the video experience are produced differently, although they do not significantly affect the fit of the model.

Indicator	**Video**	**AR**	**Reference Value**
χ^2/df	46.94 / 31 = 1.51	35.56 / 31 = 1.15	Between 1 and 3, good fit
RMSEA	0.10	0.09	$<$ 0.08 reasonable fit
RMSR	0.10	0.08	$<=$ 0.05 reflects a good fit
GFI	0.87	0.90	$>=$ 0.90 reflects a good fit
AGFI	0.76	0.83	$>=$ 0.90 reflects a good fit

Table 7.5: Indicators of structural fit model in the case of users for the video and AR experiences.

7.7 Discussion

One of the marketing company's goals in promoting and developing this AR experience was to present to its customers an image of a technologically advanced company. In this sense, regardless of the results analyzed, it can be stated that the objective was fully met and, in general, all customers enjoyed the experience to some extent, so the impact on users was very positive.

As a differentiating aspect to other works [JER18, Mol06, Pae14, FdSB16] that focus on displaying interactive 3D information looking for task efficiency, training, maintenance, or decision making, this AR experience seeks to catch the attention on the technological features of the product in potential customers or fair visitors who approached the company's stand. The simulation through AR techniques of complex mechanisms of industrial devices explains in a very didactic way how they operate. This can be done thanks to the animation of the parts that compose them, their scaling in case they are very small, and their visualization on the device in case they are hidden. This kind of simulation is essential for the demonstration of industrial products and has a great impact on the users.

From the data collected in the experiments and the SEM model proposed in this work, it can be observed that *Spatial Presence* of users has an important influence on increasing customer satisfaction with the displayed product. This is seen in the comparison of responses to the test carried out for users who could only visualize the experience through a video (see Table 7.1). Moreover, unlike other works in AR such as [HdRC$^+$17, RFH19] that seek to improve the service experience on the user individually, the AR setup proposed is a collective experience where users see themselves on the television screen and exchange opinions, being able to verify that this interaction between the users stimulated that satisfaction.

As for the hypotheses related to *Perceived Value of Experience* in terms of utility and enjoyment, it was found that, although AR technology is still far from being able to integrate virtual elements into the real world with a high degree of fidelity, participants considered the attributes of virtual objects as if they were real improving both their understanding of product functionality and their impression of product quality.

Finally, it is shown that in AR applications such as this one, there is a positive relationship between *Spatial Presence* and *Product Decision-Making*. Therefore, it

is considered of great importance the development of this type of experiences to understand the operation of complex systems in an attractive way in which users are involved as final consumers of the product.

7.8 Conclusions and Future Work

This paper presents an AR application for the display and sale of ceramic molds. For this purpose, the physical element of the ceramic mold is visualized combined with virtual components of the interior of the ceramic mold in real time. All the action is developed by an avatar that is incorporated during the visualization in AR explaining the functioning of the mold. Users visualize the graphic information about the mold, placed between them and a large-format television screen. In this way, users see themselves on the television as if it was a mirror.

In order to analyze the impact of the AR experience on users, two experiments were conducted with user tests. The first one on the AR experience and the second one on the same experience but in video format. For this purpose, this paper presents a theoretical framework that explains how users perceive and evaluate the benefits and quality of AR augmentation through their physical presence, compared to viewing the same experience through a video.

The results confirm that the use of AR played a positive role on users by enhancing their understanding of the product technology displayed by the company. The *Perceived Value of the Experience* in terms of usefulness and enjoyment was found to enhance comfort in the decision on *Purchase Intentions*. Furthermore, this stimulus was reinforced by the *Spatial Presence* of the users during the AR experience.

In this context, it can be deduced that content creation that has a clear and persuasive use is an important aspect for companies when promoting their products. Improvements in AR visualization with more realistic content and user feedback lead to an exploration of visual technology and digital design, leading to the development of new applications in other fields such as entertainment, sports and gaming.

In the future, it will be necessary to explore adjacent vias to the one exploited in this work by adding protocols for interaction with users of this type of experience, even testing its reliability in other contexts such as virtual reality (VR). Examples of these lines would be a comparison between different immersion levels such as a first-person VR experience and a screen display, or the usage of prop elements to enhance the interaction with AR experiences like the one presented for this work.

Chapter 8

Virtual Reality versus Desktop Experience in a Dangerous Goods Simulator

Publication

Chover, Miguel., Sotoca, Jose M., & Marín-Lora, Carlos. (2022, May). Virtual Reality versus Desktop Experience in a Dangerous Goods Simulator. International Journal of Serious Games, 9(2), 63–77. (Q1).
https://doi.org/10.17083/ijsg.v9i2.493.

Abstract

Virtual Reality applications have become a trend in training simulators as an alternative to desktop applications. However, further study is needed on how these types of serious games, which often include several modes of interaction, can improve the user experience. In this sense, this paper analyzes the differences between playing serious first-person games on a desktop computer versus playing in Virtual Reality. For this purpose, two versions of a dangerous goods unloading simulator have been implemented. The first one was developed as a classic desktop game with keyboard and mouse-based interaction, while the second was for Virtual Reality devices. The user experience has been measured with the In-game version of the Game Experience Questionnaire. With this, aspects related to immersion,

flow, positive emotions, and psychological needs have been compared for these two platforms. The study shows that the Virtual Reality experience produces a better overall game experience for most analyzed items. Nevertheless, the results highlight a significant dependence between the application type and the game experience induced on the player.

Keywords

Serious games, Virtual reality, Game experience

8.1 Introduction

Virtual Reality (VR) increases the reality experience felt by the user, including sensations such as touch, vision, and sound within a virtual environment created by a computer [Kim16]. Consequently, advances in the development of VR hardware devices and computer graphics technology have enabled the generation of several applications, allowing the user to enjoy spatial and temporal experiences virtually. In addition, there is a growing demand for research into technologies that support these applications, as well as an increase in total consumer spending in the video game industry using VR systems [ESA17, BC18b, HŠM+19].

VR supposes progress in terms of interaction and complete immersion of the player in the game. Moreover, VR games provide interesting advances in the contemporary video game scene. In this way, it is possible to afford innovative experiences for present and future players [Cox14]. However, the studies carried out on how VR affects gameplay are still limited, and it is not clear how interaction through VR controllers can help or harm the player's experience.

In this sense, the player's experience is more related to personal and individual enjoyment while playing with the game, which determines a subjective assessment of the quality of the game. Although we cannot establish a unique definition that defines this experience from literature, we can indicate that several elements can influence the feelings and experiences that people have when they play digital games such as enjoyment, immersion, challenge, etc [GKN11].

In the literature, several questionnaires have been developed that offer different significant elements in the player's experience [NDC14, DNC16]. Among them we can highlight, the Game User Experience Satisfaction Scale (GUESS) [PKC16] that

performs an exhaustive study of the different aspects that influence the development of a video game, the Core Elements of the Gaming Experience questionnaire (CEGE)[CGCC15] in which aspects such as enjoyment while playing are considered, the interaction formed by the player's sense of control and ownership, and the video game itself formed by the environment and the gameplay, and the Game Engagement Questionnaire [BFC+09] that analyzes what aspects are associated with the negative effects of violent video games.

This paper evaluates a serious game through VR by comparing it to viewing the same game in First Person (FP) on a desktop display. Therefore, and as previous work in the learning process evaluation of this type of games [BKLMG13], it is proposed to carry out an analysis on the player's experience that allows evaluating which technology could be more useful in this framework. In this sense, and following the work of Pallavicini & Pepe [PP19], we will use the Game Experience Questionnaire (GEQ) [IdP13] to compare both technologies, and more specifically the *In-game GEQ* version. This questionnaire measures the following seven components: *Competence*, *Sensory and Imaginative Immersion*, *Flow*, *Tension*, *Challenge*, *Negative Affections*, and *Positive Affections*.

Through competence, an attempt is made to measure the intrinsic motivation that players feel when it comes to fulfilling the requirements of the tasks they wish to complete [GKN11]. Emri and Mäyrä [ME11] studied immersion in the game as part of the player's experience, proposing a model that includes three different aspects in the immersion process: sensory, challenge-based, and imaginative immersion. Referring to sensory immersion as the multisensory properties of a game, in other words, the characteristics of the game that generate a perceptual impact on the user. Challenge-based immersion involves analyzing the cognitive aspects necessary to overcome the game challenges, while imaginative immersion refers to the fantasy created in the game, and depends on the richness of the narrative structure. In the case of VR, this technology supposes a more intense degree of immersion compared to a computer screen, making the user consider the virtual world of the game that surrounds him/her as the real world.

About the *Flow* dimension, we can define it as the sensation of influencing the game activity within the virtual world, and in that sense, we can consider it one of the important aspects involved in the player's enjoyment [WWH+08]. Another dimension to consider is the *Positive* or *Negative Affections* related to the emotions that are generated in the player during the game. Positive psychological aspects such as

happiness or surprise can increase the success of the game [Tan08]. Nevertheless, when the challenge is unbalanced in its complexity, the player may experience negative emotions, including tension due to poor ability to solve tasks or discomfort if the game offers little difficulty, losing interest in continuing to play [SW14].

The reason for using the *In-game GEQ* versus the other different versions proposed in [IdP13] is that the *GEQ - Core Module*, consisting of 33 items that probe players' feelings and thoughts while playing, has been questioned in the [LBM18] work. In Law et al. [LBM18], the authors conclude that some items were inconsistent after measuring psychometric properties in the gaming experience of 633 participants after they had played in the past 24 hours. For this purpose, the authors performed an Exploratory Factor Analysis (EFA) on the seven factors to indicate whether the items correspond to the seven components indicated in the questionnaire. Furthermore, it also emerges from the study that the *Tension* and *Negative Affections* components are too similar and should be merged into a single component.

It is noteworthy that among the questions in which they find inconsistencies (see Table 5 in [LBM18]) only question 5 appears in the *In-game GEQ*, with the difference that in the *GEQ - Core Module* it is defined as "I was fully occupied with the game" while the *In-game GEQ* is the only one that has been modified in this work and is defined as "I felt completely absorbed". Moreover, at no point is a solution shown to the inconsistencies found in some [IdP13] items. Therefore, for the present work we have chosen the *In-game GEQ* constructed by 14 items.

As a summary, this document is organized as follows. Section 8.2 presents the references that have been used for the study of previous work in the fields related to the design and implementation of this experience. Next, section 8.3 details the context in which the application is carried out and continues with the necessary elements for the development of the serious game. Later, in section 8.4, the hypotheses and general objectives of the study carried out are presented. Sections 8.5 and 8.6 show the results and a discussion of them. Finally, section 8.7 outlines the conclusions obtained from this work and the possible lines of future work.

8.2 Literature on the State of the Art

In recent years, several works have been carried out to address immersive VR, taking into account different aspects such as interaction, the user interface, the

haptic system, or the player's movement in the environment [CB20, BV17]. All this allows the user to know where he/she is, with whom he/she interacts, and what actions to perform. In this way, the user perceives the virtual world as a reality where he/she can interact by adding haptic devices and audio sources that maximize spatial presence [CNdSR16, SNM16].

Currently, last generation devices include VR headsets such as the *Oculus Quest 2.0* or the *HTC Vive Pro 2.0*. At a more affordable level are the *Samsung Gear VR* and *Google Cardboard* which work by using a smartphone as a display. This has made it easier for consumers who want to experiment with visual immersion to use these types of technologies.

To provide more realistic interactions between the virtual and real worlds, several technological developments have been made. One of them is based on visual satisfaction such as gaze-based hand interaction using *Oculus Quest 2.0* or leap motion device, to represent realistic movements and gesture recognition and analysis [HK17, VAKAM20]. Furthermore, the user's immersion can be enhanced by adding touch processing through a haptic device to enable feedback of physical reactions occurring in a virtual environment or user-to-user interaction [LSGF$^+$19, COB$^+$18, KHF$^+$19, PCPM13]. About touch accuracy, Leonardis et al. [LSBF17] include a three revolute-spherical-revolute (3-RSR) haptic wearable device which allows control of the contact of the fingertips. This is a new three degrees-of-freedom wearable haptic interface that uses force vectors directly on the fingers.

Another aspect to be considered is the specification of displacement or locomotion, which requires providing the user with a way to control their movement in the world. Locomotion in VR involves traveling in a virtual world of infinite scale while remaining in the confines of a real-world at the scale of the room in which the user is located [VKBS13, AS18]. There are several possible techniques to solve the problem of locomotion [BRKD19], with different usability characteristics [HZQ$^+$19]. Among the most important strategies, we can mention: the use of game controls or joysticks, teleportation, or controller movement [BC21], head motion sensors [TAZF17] or the establishment of reference points [JHMWA18]. All this is complemented by wayfinding in a virtual environment, that is, the ability to determine a route, learn it, and go back over it or reverse from memory [Gol99]. The virtual environment is often unfamiliar to new users, and therefore it is essential to provide tools to orient themselves [BK15]. In this sense, the spatial structure of the environment can influence the purposeful and directed movement based on the objectives pursued

[YDOT+19].

However, it is not clear the advantages and disadvantages that VR brings us through a head-mounted display in comparison with viewing the same experience on a desktop screen [JKK+20, PP19]. While some works indicate that a FP Desktop system visualization implies a higher performance and usability concerning VR [MSG+15, TLS+15], other investigations show that these differences cannot be considered significant [PPM19].

In particular, although a greater intensity has been demonstrated in terms of immersion and presence in VR games compared to desktop games, it does not mean that its use is ideal for all types of games. In the case of driving simulators, it was seen that VR technology is not the best solution, preferring a flat-screen condition, where the participants were seated in front of three flat screens with a combined resolution of 5760 x 1080 pixels and a field of view (FoV) of approximately 135° depending on the size of the participants [WFR+17].

8.3 Serious Game Description

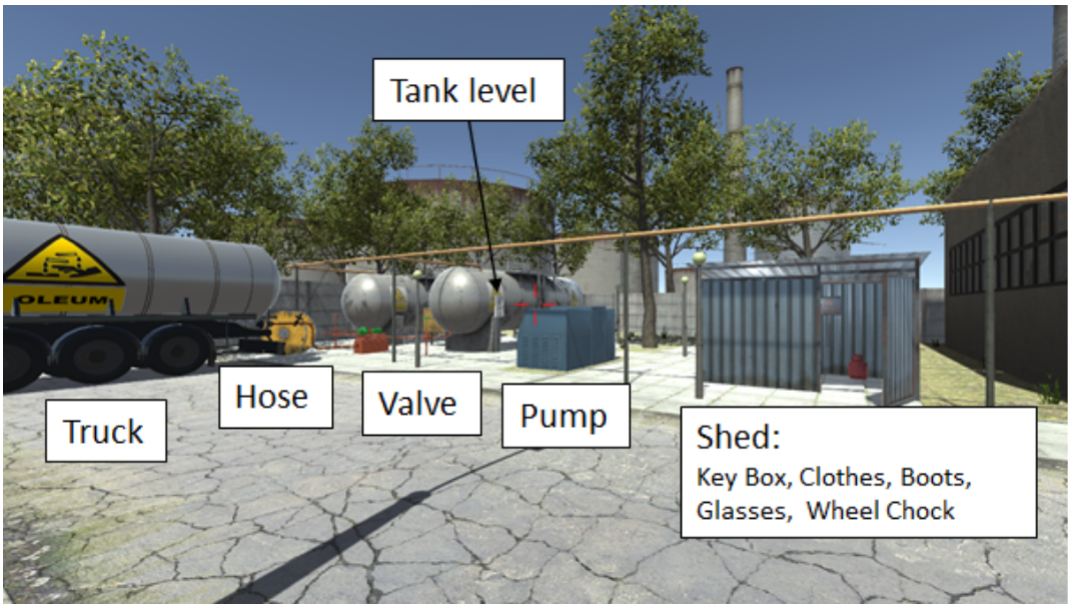

Figure 8.1: Virtual Simulator for Learning Dangerous Goods Operations

The simulator developed allows training the workers of a chemical company in the unloading operations of dangerous goods. The environment includes the necessary

elements to perform the usual tasks in these types of operations. First of all, there is a shed where the safety equipment is kept, the wheel chock to stop the truck and the box to keep the keys while the unloading operation is being carried out. In addition, the environment has a representation of the truck and the two unloading tanks. There are also elements to interact with such as the hose, the unloading valves, the pump, and the truck driver. Figure 8.1 shows a screenshot of the scene. The application has been developed to be able to design different unloading conditions. In this respect, it is possible to choose the type of substance to be unloaded (oleum or caustic soda), the filling level of the unloading tank (empty or full) as well as the atmospheric conditions to simulate the unloading with or without rain.

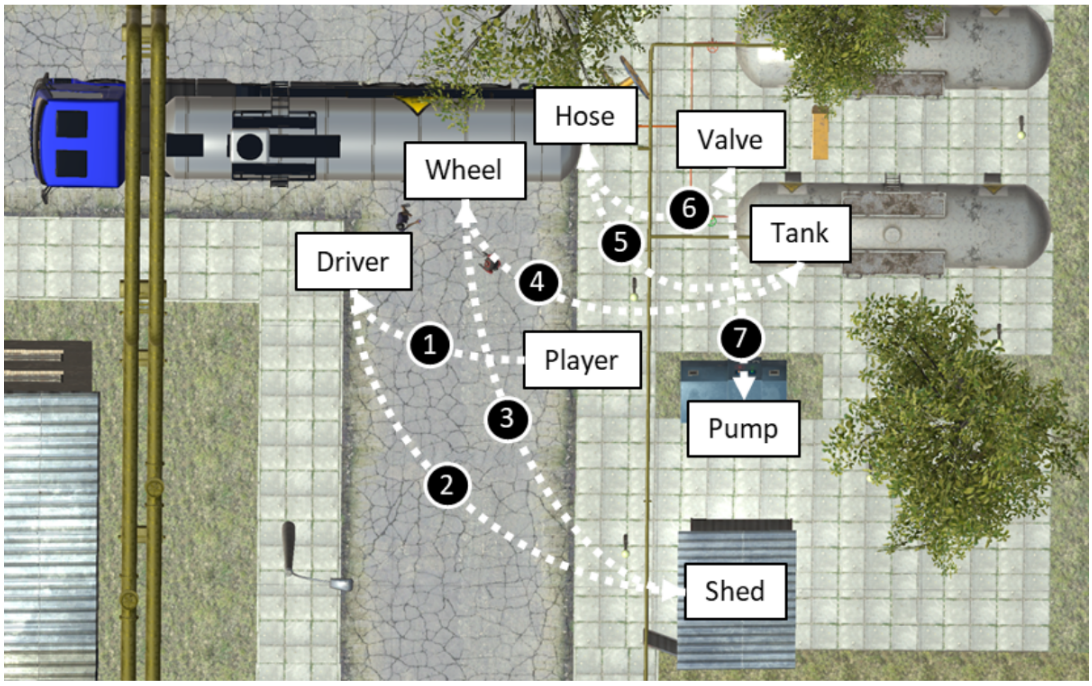

Figure 8.2: Sequence of actions map

The sequence of actions to perform the assigned tasks involves the locomotion or displacement of the operator as shown in the floor diagram (see Figure 8.2). This locomotion is performed without teleportation. The sequence is made up of seven routes. First, the player must go to the driver (route 1), then to the shed (route 2), after this he/she returns to the truck to place the chock on the wheel (route 3), then goes to check the tank level (route 4) and to place the hose (route 5), then the player must go to the valve (route 6) and finally turns on the pump (route 7). After this,

the process is performed in reverse to undo all actions and finally return the keys to the driver to end the experience. During its execution, the system keeps track of the actions performed by the user and displays error messages if the correct sequence is not performed, indicating which of the listed actions will be the next to be performed. Finally, the application generates a report to validate the operator's performance.

(a) Take the keys from the driver

(b) Carry the wheel chock

(c) Checking if the tank is full

(d) Press the button to start the pump

(e) Disconnect the hose from the truck

(f) Close the oleum valve

Figure 8.3: Interaction examples in the simulator

The system was developed to allow different types of interaction with environmental elements (see Figure 8.3), beyond moving around the stage. For example,

some objects can be picked up, carried, and deposited, for instance, the driver's keys (Figure 8.3(a)) and the wheel chock (Figure 8.3(b)). Other objects can simply be picked up, such as clothing, boots, or glasses, and others can be pressed, including the tank emptying (Figure 8.3(c)) and pump start/stop buttons (Figure 8.3(d)). Finally, two-handed interaction is possible when connecting and disconnecting the hose (Figure 8.3(e)) or turning the valves (Figure 8.3(f)).

Starting from the initial configuration generated with the variables defined for the experience (type of load, filling level of the receiving tank, and weather conditions), the operator must complete the following actions, performing when necessary the displacements indicated in the paths shown in Figure 8.2:

1. Wait for truck entry and parking for unloading and then go to the driver (route 1).
2. Take the keys from the driver after he/she gets off the truck.
3. Take the keys to the key box in the shed (route 2).
4. Dress according to the type of load.
5. Put on non-skid boots if it rains.
6. Put on safety glasses.
7. Take the wheel chock to secure the truck.
8. Carry the wheel chock and place it on the rear wheel of the truck (route 3).
9. Go to the tank to check its level and empty the discharge tank if it is full (route 4).
10. Go to the discharge hose to connect it (route 5).
11. Go to the corresponding valve depending on the type of product to open it (route 6).
12. Go to the corresponding pump and press the start button (route 7).
13. Wait for the end-of-load sound signal.
14. Press the button to stop the corresponding pump.

15. Close the valve that has been opened (reverse route 7).
16. Disconnect the hose from the truck (route 6).
17. Remove the wheel chock that brakes the truck (reverse routes 5 and 4, it is not necessary to check the tank level after unloading).
18. Carry the wheel chock to the shed (reverse route 3).
19. Take the keys from the key box in the shed (reverse route 2).
20. Carry the keys and return them to the driver (reverse route 1).

To test the differences in terms of gaming experience of a simulator developed for desktop computers versus the one developed for VR, two versions of the simulator have been developed. Both versions have the same functionalities and simply differ in the interaction and display devices. The interaction in the Desktop application is done with keyboard and mouse and the visualization is on a computer monitor, while in the VR application the interaction is done with the controllers, and the scene is seen through the helmet. It also changes the position of the player, who is sitting in the desktop version and standing in the VR version (see Figure 8.4).

Figure 8.4: Players testing both versions of the serious game

8.4 Experiments Description and Scope

Nowadays, VR serious games have proliferated due to their unique immersive and interactive features. These video games are used in the industry as an instructional tool. However, there is no scientific evidence to justify their use against games

developed for conventional desktop display devices [PP19]. For this reason and as a previous work to the evaluation of learning outcomes [BKLMG13], it is proposed to conduct a study on player experience to initially assess which technology might be most useful in this context. In this sense, and following the work of Pallavicini & Pepe [PP19], the *In-game GEQ* [IdP13] will be used to compare both technologies.

For this purpose, some key aspects such as the immersion level, fluency, positive or negative emotions, challenge, competition, and tension/anxiety, will be analyzed with the same game played on a desktop screen and in VR. The main hypotheses of this study were:

- **H1**. The dangerous goods serious game played in VR produces more positive emotions than the Desktop one.
- **H2**. Immersion and flow are more intense in the VR version than in Desktop version.
- **H3**. Differences in psychological needs (i.e., sense of competence, tension/annoyance, and challenge) could be relevant between the two technologies for this experiment.

Concerning the design of the experiments, the comparison conditions for each of the experiments were as follows:

- **Simulation in Desktop**: Participants were seated in front of a 27-inch *iMac* personal computer with the Desktop simulator version running in full screen. User interaction with the simulator was performed with keyboard and mouse as is typically done in these games.
- **Simulation in VR**: Participants put on the VR headset *Oculus Quest 2* with the VR simulator version running. The interaction was done with the VR controllers provided by the system.

About both games' implementations, the Desktop version of the simulator was developed with *Unity 3D 2019.2.2* and the *Kineractive 1.11* plugin that allows the creation of complex reverse kinematic interaction. And the VR version of the simulator was developed for *Oculus Quest 2* with *Unity 3D 2019.2.2* and the *HurricaneVR 2.3* plugin which consists of a physical interaction toolkit that allows the creation of

immersive VR games. The software used for the statistical analysis was *Matlab R2018b*.

For the experiment's procedure, the experiment was carried out by 60 participants, 31 women and 29 men, with a mean age of 23 years (Standard Deviation = 7.3; minimum age 18 years, maximum age = 56 years). The only condition to participate in the study was that the participants did not have any significant visual impairment (all have normal or corrected to normal visual acuity). The study has received the approval of the Ethics Committee of the Jaume I University of Castellón. The participants were scheduled in pairs and in 20 minutes time slots. One of them had to pick one of the two experiences at will, and the other would go directly to the other. In both experiences, users had a few minutes to get used to the application environment and interaction mechanisms. Once ready, users played with the game for 10 to 15 minutes. For the simulation to be carried out successfully, participants had to perform the sequence of actions presented in Figure 8.2 and outlined in Section 8.3. Since the participants had no previous training in dangerous goods unloading, an assistant guided them and explained the steps to be taken in case of doubt. At the end of the experience, they filled out a questionnaire about their game experience.

The questionnaire chosen for both experiences was the GEQ [IdP13] using its *In-game GEQ* version. This questionnaire consists of 14 items for users to express their impressions, rating each item on a five-point Likert scale ("very unfavorable" = 0 to "very favorable" = 4). The *In-game GEQ* collects the following seven different components and two items are used for every component. The items for each are listed below: *Competence* (items 2 and 9), *Sensory and Imaginative Immersion* (items 1 and 4), *Flow* (items 5 and 10), *Tension* (items 6 and 8), *Challenge* (items 12 and 13), *Negative Affections* (items 3 and 7) and *Positive Affections* (items 11 and 14). In addition, item 1 has been slightly modified from "I was interested in the game's story" to "I was interested in the operations sequence of the game" as it deals with actions on a serious game of dangerous goods. Table 8.1 shows the statement of the *In-game GEQ* questions associated with their corresponding components.

8.5 Results

After collecting and analyzing the data collected in the questionnaires, the results obtained from the experiments are presented in the following. To analyze the degree

of the normativity of the different items, a Lilliefors test was performed for the Desktop and VR simulators, calculating their statistical value and their p-value. To compute the critical values for the hypothesis test, interpolated values are calculated on a table of previously calculated critical values using Monte Carlo simulation for sample sizes less than 1000 and significance levels between 0.001 and 0.50. The cutoff value with this statistic for 60 samples is 0.114 for a 5% level test. For all items the Lilliefors test statistic is greater than the cutoff value, so we reject the normality hypothesis. Consequently, an ANOVA test does not demonstrate the statistical significance of the responses to the questionnaire for the two simulators.

In this work, two non-parametric statistical tests have been used to determine the statistical significance of the results: the Kruskal-Wallis and Friedman test. Kruskal-Wallis test is a non-parametric version of classical one-way ANOVA, and an extension of the Wilcoxon rank-sum test to more than two groups. It compares the medians of the data groups to determine if the samples come from the same population. The approach uses data ranks, rather than numeric values, ordering the data from least to greatest in all groups and calculating the sum. Friedman's test is similar to classical balanced two-way ANOVA. This approach compares the means using data ranks. In both statistical tests use the Chi-squared statistic and the p-value. The criterion to reject the null hypothesis at the 5% significance level will be when the p-value > 0.05.

Competence	Item 2. I felt successful Item 9. I felt skilful
Sensory and Imaginative Immersion	Item 1. I was interested in the operations sequence of the game Item 4. I found it impressive
Flow	Item 5. I forgot everything around me Item 10. I felt completely absorbed
Tension	Item 6. I felt frustrated Item 8. I felt irritable
Challenge	Item 12. I felt challenged Item 13. I had to put a lot of effort into it
Negative Affections	Item 3. I felt bored Item 7. I felt it tiresome
Positive Affections	Item 11. I felt content Item 14. I felt good

Table 8.1: List of the items in the *In-game GEQ*.

Table 8.2 shows the items associated with their components for both simulators (Desktop and VR), where column 4 shows their mean and standard deviation and where bold values indicate the simulator with the highest score. As for the components related to *Tension* and *Negative Affections*, it should be noted that the score interpretation for these two components is different, being better when lower values are obtained in them. Columns 5 and 6 show the statistical significance of the

Components	**Item**	**Simulator**	**Average± Std Deviation**	**Kruskal-Wallis Test**	**Friedman Test**
Competence	2 *	Desktop	2.55 ±1.17	$\chi^2(1)$ = 7.305	$\chi^2(1)$ = 7.989
		VR	**3.12 ±0.87**	p=0.007	p=0.005
	9 *	Desktop	2.00 ±1.19	$\chi^2(1)$ = 13.695	$\chi^2(1)$ = 14.291
		VR	**2.78 ±0.88**	p<0.001	p<0.001
Sensory and Imaginative Immersion	1 *	Desktop	2.28 ±1.11	$\chi^2(1)$ = 26.340	$\chi^2(1)$ = 30.613
		VR	**3.28 ±0.99**	p<0.001	p<0.001
	4 *	Desktop	1.91 ±1.14	$\chi^2(1)$ = 13.695	$\chi^2(1)$ = 14.291
		VR	**3.17 ±0.96**	p<0.001	p<0.001
Flow	5 *	Desktop	1.67 ±1.35	$\chi^2(1)$ = 31.466	$\chi^2(1)$ = 40.500
		VR	**3.10 ±1.08**	p<0.001	p<0.001
	10 *	Desktop	1.33 ±1.17	$\chi^2(1)$ = 40.473	$\chi^2(1)$ = 43.667
		VR	**3.00 ±1.21**	p<0.001	p<0.001
Tension	6	Desktop	0.50 ±0.87	$\chi^2(1)$ = 3.472	$\chi^2(1)$ = 5.558
		VR	**0.22 ±0.52**	p=0.062	p=0.018
	8	**Desktop**	**0.10 ±0.30**	$\chi^2(1)$ = 0.350	$\chi^2(1)$ = 0.236
		VR	0.15 ±0.66	p=0.554	p=0.626
Challenge	12 *	Desktop	1.45 ±1.31	$\chi^2(1)$ = 9.636	$\chi^2(1)$ = 11.757
		VR	**2.22 ±1.25**	p=0.002	p<0.001
	13 *	Desktop	0.83 ±0.89	$\chi^2(1)$ = 12.421	$\chi^2(1)$ = 14.340
		VR	**1.37 ±0.74**	p=0.001	p<0.001
Negative Affections	3 *	Desktop	0.93 ±1.19	$\chi^2(1)$ = 18.002	$\chi^2(1)$ = 24.667
		VR	**0.20 ±0.55**	p<0.001	p<0.001
	7	Desktop	0.35 ±0.78	$\chi^2(1)$ = 1.402	$\chi^2(1)$ = 1.401
		VR	**0.20 ±0.55**	p=0.236	p=0.236
Positive Affections	11 *	Desktop	2.40 ±1.11	$\chi^2(1)$ = 22.407	$\chi^2(1)$ = 32.236
		VR	**3.30 ±0.91**	p<0.001	p<0.001
	14 *	Desktop	2.72 ±1.11	$\chi^2(1)$ = 15.999	$\chi^2(1)$ = 22.801
		VR	**3.43 ±0.85**	p<0.001	p<0.001

Table 8.2: Average score, standard deviation and statistical significance for questionnaires.

Kruskal-Wallis and Friedman tests indicating their Chi-squared and p-value. Based on this information, if one of the two statistics is not significant, it is considered that the item does not have sufficient statistical significance and, therefore, only items with statistically significant results are marked with an asterisk in column 2.

Overall, the results presented in Table 8.2 reveal that participants' ratings for the different items are better when playing the VR experience than when playing on the Desktop mode. These differences are seen in the components of *Competence*, *Sensory and Imaginative Immersion*, *Flow*, *Challenge*, and *Positive Affections*. In the case of the item related to *Tension*, there is no significant difference between the two simulators, producing in both cases a low-stress level. As for the *Negative Affections*, there is no significant difference about item 7 "I felt it tiresome", while for item 3 "I felt bored" the VR simulator scores better, probably due to the "wow effect" when using this type of technology.

8.6 Discussion

8.6.1 Positive Emotions

Analyzing the results about the starting hypotheses, the following conclusions can be drawn. Firstly for hypothesis H1, the participants' impressions of positive emotions show that the VR experience makes them feel better. There is a clear difference in the average values obtained for both simulators. In addition, these results have statistical significance as can be seen in the values obtained for items 11 and 14 in Table 8.2. This perception has been demonstrated by other research works, with some exception [WFR+17]. Although, the most widespread assumption is that VR causes the so-called "wow effect" [RPB17] that produces such positive emotions.

8.6.2 Immersion and Flow

Furthermore, it can be seen from the results that there is a clear difference in the preference of the participants in the questionnaire in favor of VR simulation over Desktop in terms of the components related to immersion and flow. This confirms hypothesis H2 regarding how VR impresses, isolates, and absorbs users from the outside world. Furthermore, the results shown in Table 8.2 indicate that there is a

large difference between the scores obtained on average for these components, which also show statistical significance for both Kruskal-Wallis and Friedman tests.

8.6.3 Psychological Needs

Finally, regarding the psychological needs, it should be noted that hypothesis H3 establishes that the differences between the two simulators could be significant (in contrast with other authors [PP19, WFR+17]). In this sense, the results show clear differences in favor of the VR experience regarding *Competence* and *Challenge*, and all the items involved show statistical significance. However, about *Tension*, the differences are less visible, and only item 6 "I felt frustrated" which is slightly lower in the case of VR is close to statistical significance. Moreover, if *Negative Affections* are analyzed only item 3 "I felt bored" is lower in the VR experience, while item 7 "I felt frustrated" is very similar.

8.6.4 Other Considerations

Because of the results obtained and taking into account that the experiences are different from others studied in the literature [PP19, PPM19], it can be concluded that the dangerous goods unloading simulator does present significant differences in terms of the psychological needs of the participants when they play in Desktop vs VR. About *Competence*, users felt more capable and skilled in the VR experience. In terms of *Challenge*, the VR experience was more challenging and thought-provoking. On the other hand, regarding *Tension*, although the differences are less significant the VR experience provoked slightly less frustration although similar irritability among participants.

It should be noted in any case, that in the proposed experience the level of difficulty for the user is higher since there are tasks in the dangerous goods unloading simulator that require complex actions. In addition, users have to move around a virtual environment to perform different tasks.

8.7 Conclusions and Future Work

The continuous advances in the development of VR technologies and their application in training make it necessary to study the advantages of these applications over traditional desktop solutions. In the same way, it is also worthwhile to assess the

new forms of interaction that these new technologies provide and their application in specific fields.

In this regard, the developed work performs an analysis to evaluate the game experience in a simulator for learning the tasks of unloading trucks carrying dangerous goods. The study compares two versions of the simulator, the first one running on a Desktop computer and the second one as a VR application. The study shows that the VR experience produces better overall results for most of the components in the *In-game GEQ*. The study results suggest that there are significant differences in the psychological needs of the participants, mainly in terms of *Competence* and *Challenge*. However, with frustration, irritability, and tiresomeness the feeling is similar in both simulators.

However, it seems apparent that there is still research work in this field. There is a dependency between the application type and the game experience raised by the player from the literature [PP19, CB20]. It is not the same to play a game sitting or standing, or with a proxy device such as a steering wheel and VR controllers or hands.

In the future, to verify these guesses and analyze the advantages and disadvantages of using different technologies, it would be necessary to experiment with many more applications that address the problem from different perspectives. For instance, using alternative interaction systems or different physical objects (proxy objects). Another aspect is the user's success and performance analysis. In cases such as dangerous goods handling, the experience has to fulfill its purpose and provides knowledge and practice to improve their working conditions and safety.

Part V

Afterword

Chapter 9

Conclusions and Future Work

Today, the tools for developing video games have a strong technical dependence and require some knowledge for their use. In this sense, the aim is to reduce their use complexity and facilitate access to the creation of video games. For this purpose, it starts with the essential elements that define games and designing and developing a game engine that reduces their complexity of use without affecting their potential. Next, the game engine is formalized so that the produced games meet the characteristics of multi-agent systems, and a method is defined for the game specification as a multi-agent system independently of the development platform. Finally, based on the implementation of two virtual reality and augmented reality applications, the game experience in serious games specified according to the proposed multi-agent methodology is studied. This chapter presents the general conclusions and the lines of future work derived from the contributions of this thesis.

9.1 Conclusions

In a similar way to the contributions that make up this thesis, the conclusions are structured in three main blocks. The specific conclusions of each block are presented below.

9.1.1 Design and Development of a Game Engine

The contributions made in this block cover the study of the specification of video games, emphasizing the mechanisms that define and compose the game logic by including non-redundant or accessory elements to the model. The purpose behind this

restriction is twofold. Firstly, propose an alternative to the traditional development methods of these interactive applications that usually include advanced functions. Secondly, obtain a game specification that does not rely on the data structures of any programming language. This approach results in a game specification model based on sets of *Scenes* composed of a list of *Actors* with generic properties and methods and a reduced set of actions and conditions to define behaviors.

The result is a game development environment where there are no hierarchies or scene graphs and where there are no dependencies on arrays, loops, and other complex data structures. For the validation of this result several tests and experiments have been carried out with users to verify its usefulness and ease of use compared to other game creation environments. The data obtained from these experiences show that users can develop games more simply compared to other commercial engines. It seems significant that all participants completed their game implementations despite having no previous experience. Furthermore, from the comparative study against another visual programming environment, it is obtained that the proposed model is perceived as easier to use and understand its operation.

9.1.2 Game Engine Formalization Using Multi-agent Systems

The general goal of this block is to specify the analogy between video games and multi-agent systems for the specification and implementation of games. The result of this work keeps the structure of the game engine, where the entities that compose the games share a property set and a generic logic system, and where there are no hierarchical dependencies.

The validation of the method has been carried out through the specification and the implementation of three games. The criteria used to select these games leans on the observation that they all present common game mechanics. This leads to the deduction that the characteristics of the multi-agent systems fit the definition of video games. The results obtained from the development of these games validate the theoretical formalization of the video game development environment and show the potential of the game engine to create games as multi-agent systems.

Additionally, derived studies and experiments have been used to explore complementary pathways. From these works, it can be deduced that the incorporation of agent specification systems in the development of video games facilitates the understanding of them before their implementation, which contributes to the access

to the sector to further professional profiles.

In addition, these results show that the proposed model allows the specification and implementation of video games regardless of the platform for their development. This process has been validated using arcade games as use cases with game mechanics widely extended. These games have been specified and successfully implemented in various development environments, demonstrating that the proposal meets its initial objectives and opens the door to its application in contexts other than arcade games.

9.1.3 Serious Games Development

Finally, the third block applies the previously described multi-agent specification model to serious game development experiences in any game engine. The purpose is to check if the framework allows specifying games with more complex characteristics than arcade games in a way independent of the game engine where the game will be developed.

More specifically, the two works developed in this block are virtual reality and augmented reality serious games implemented on a commercial game engine. The virtual reality game is for training the tasks of unloading trucks transporting dangerous goods through virtual reality and the augmented reality game is for the visualization and sale of ceramic molds.

Both are specified using the proposed formal method, and the result is two fully functional applications that fulfill their purpose from a definition and a multi-agent specification as proposed.

The development of these two games had general goals beyond the specification system as the evaluation of game experiences. In the case of the virtual reality game, the main interest of the study lies in the interaction mechanisms, where some objects are picked up, carried, and deposited, others can be picked up or clicked, and others require two-handed interaction. From this study, it is obtained that in virtual reality developments it is desirable to define and evaluate new forms of interaction. In addition, it seems necessary to continue studying the methods of definition, visualization, and interaction, since a dependence between the type of application and the game experience is detected. The augmented reality serious game, meanwhile, consists of the projection of synthetic elements on a ceramic mold through augmented reality in the context of a trade fair. In this case, the study

evaluates the gaming experience on how users perceive the benefits and quality of the game through its physical presence, compared to viewing the same experience through a video. The gathered results reinforce the positive role of augmented reality through parameters such as the perceived value of the experience, and comfort in the purchase intention decision enhanced by spatial presence.

In summary, it can be deduced that creating realistic and interactive content is key for companies when promoting their products. This supports one of the premises of this work by indicating that it would be interesting to extend the development scope to other profiles that can create applications in fields such as marketing, entertainment, or sports.

9.2 Future Works

Throughout the development of this work, multiple ideas have arisen to continue and extend the results obtained in terms of video game development and game experience.

One of them explores the game engine's capabilities and the multi-agent specification system with its extension to 3D. Currently, it is already possible to create 3D environments or simple games with editors such as *Minecraft* or *Kodu*, but as stated in Chapter 1, it is still necessary to have advanced knowledge to create games in development environments such as *Unity*, *Unreal* or *Godot*. In this sense, the 3D version of the game engine requires an analysis of the current 3D engine features and the differences from the proposed 2D model. This study aims to generate a 3D version of the game engine that keeps user-friendly and oriented to game development for non-developer profiles.

Besides, the multi-agent approach to video games needs to take a further step in its formalization. Currently, scripting in video games handles specific events and behaviors to modify game logic with no recompiling. Although many engines have proprietary languages, the trend has shifted towards the use of generic scripting languages. This means that many commercial games are developed using languages such as *Lua*, *Python*, *Ruby*, *JavaScript*, or *ActionScript*, as well as visual scripting languages. A proposal in this sense is to define a scripting language oriented to video games based on the model proposed in this thesis and on the syntax definition and semantics from the structured programming studies.

Another research line that applies the multi-agent model to video games con-

sists of the definition and implementation of algorithms and games more complex than the arcade games developed in this work. For instance, sorting or pathfinding algorithms whose traditional implementations require iterative loops and data structures to represent the space and store information about the potential resolution paths. In addition, it is also proposed the development of other games of the *Real-Time Strategy* genre or games like *Sudoku*, where cooperation and consensus techniques and even backtracking solutions are applied.

Lastly, another research area derived from this work explores alternative interaction methods and also the game experience assessment. In some of the works that make up this thesis, the first approaches to serious games in fields such as virtual reality and augmented reality have already been proposed. These investigations have shown the need to extend the proposed model for the creation of virtual and augmented reality content. It would be necessary to test and analyze the advantages and disadvantages of using different interaction technologies by experimenting with applications from various domains that approach the problem from different perspectives. An example would use alternative interaction systems or physical objects such as proxy objects or fitness devices such as treadmills or exercise bikes. In this sense, and given that there is a growing interest in applying gaming techniques to non-gaming purposes in areas such as health, training, and rehabilitation, work is currently underway on a video game for treadmill running in a 3D environment. This development integrates research on procedural generation of terrains and ecosystems with vegetation, automatic simplification and interactive visualization of trees and plants, specification of behaviors based on multi-agent systems, and design of new interaction systems for sports machines.

Bibliography

[AA17] Murat Akçayır and Gökçe Akçayır. Advantages and challenges associated with augmented reality for education: A systematic review of the literature. *Educational Research Review*, 20:1–11, 2017.

[AC19] John Aliprantis and George Caridakis. A survey of augmented reality applications in cultural heritage. *International Journal of Computational Methods in Heritage Science (IJCMHS)*, 3(2):118–147, 2019.

[ACB08] Gustavo Aranda, Carlos Carrascosa, and Vicente Botti. Characterizing massively multiplayer online games as multi-agent systems. In *International Workshop on Hybrid Artificial Intelligence Systems*, pages 507–514. Springer, 2008.

[AEMC08] Eike F Anderson, Steffen Engel, Leigh McLoughlin, and Peter Comninos. The case for research in game engine architecture. *ACM*, 2008.

[AMS+01] Rogelio Adobbati, Andrew N Marshall, Andrew Scholer, Sheila Tejada, Gal A Kaminka, Steven Schaffer, and Chris Sollitto. Gamebots: A 3d virtual world test-bed for multi-agent research. In *Proceedings of the second international workshop on Infrastructure for Agents, MAS, and Scalable MAS*, volume 5, page 6. Montreal, Canada, 2001.

[AMW+13] Eike Falk Anderson, Leigh McLoughlin, Joe Watson, Sam Holmes, Peter Jones, Hayden Pallett, and Brendan Smith. Choosing the infrastructure for entertainment and serious computer games-a whiteroom benchmark for game engine selection. In *2013 5th international conference on games and virtual worlds for serious applications (VS-GAMES)*, pages 1–8. IEEE, 2013.

[And11] Eike Falk Anderson. A classification of scripting systems for entertainment and serious computer games. In *2011 Third International Conference on Games and Virtual Worlds for Serious Applications*, pages 47–54. IEEE, 2011.

[AS10] Apostolos Ampatzoglou and Ioannis Stamelos. Software engineering research for computer games: A systematic review. *Information and Soft Tech*, 52(9):888–901, 2010.

[AS18] Jeremy Albert and Kelvin Sung. User-centric classification of virtual reality locomotion. In *Proceedings of the 24th ACM Symposium on Virtual Reality Software and Technology*, VRST '18, New York, NY, USA, 2018.

[ATE+12] Gustavo Aranda, Tomas Trescak, Marc Esteva, Inmaculada Rodriguez, and Carlos Carrascosa. Massively multiplayer online games developed with agents. In *Transactions on edutainment vii*, pages 129–138. 2012.

[Azu97] Ronald Azuma. A survey of augmented reality. *Presence: teleoperators & virtual environments*, 6(4):355–385, 1997.

[B+96] John Brooke et al. SUS - A quick and dirty usability scale. *Usability evaluation in industry*, 189(194):4–7, 1996.

[BARHN14] Christian Becker-Asano, Felix Ruzzoli, Christoph Hölscher, and Bernhard Nebel. A multi-agent system based on unity 4 for virtual perception and wayfinding. *Transportation Research Procedia*, 2:452–455, 2014.

[BBA+01] Olivier Barreteau, François Bousquet, Jean-Marie Attonaty, et al. Role-playing games for opening the black box of multi-agent systems: method and lessons of its application to senegal river valley irrigated systems. *Journal of artificial societies and social simulation*, 4(2):5, 2001.

[BBS19] Kamal Berahmand, Asgarali Bouyer, and Negin Samadi. A new local and multidimensional ranking measure to detect spreaders in social networks. *Computing*, 101(11):1711–1733, 2019.

[BC18a] Marie Beck and Dominique Crié. I virtually try it… i want it! virtual fitting room: A tool to increase on-line and off-line exploratory behavior, patronage and purchase intentions. *Journal of Retailing and Consumer Services*, 40:279–286, 2018.

[BC18b] Fabio Buttussi and Luca Chittaro. Effects of different types of virtual reality display on presence and learning in a safety training scenario. *IEEE Transactions on Visualization and Computer Graphics*, 24(2):1063–1076, 2018.

[BC21] Fabio Buttussi and Luca Chittaro. Locomotion in place in virtual reality: A comparative evaluation of joystick, teleport, and leaning. *IEEE Transactions on Visualization and Computer Graphics*, 27(1):125–136, 2021.

[BCJS99] Vicente Botti, Carlos Carrascosa, Vicente Julián, and Jose Soler. Modelling agents in hard real-time environments. In *European Workshop on Modelling Autonomous Agents in a Multi-Agent World*, pages 63–76. Springer, 1999.

[BCL15] Mark Billinghurst, Adrian Clark, and Gun Lee. A survey of augmented reality. 2015.

[BFC+09] Jeanne H. Brockmyer, Christine M. Fox, Kathleen A. Curtiss, Evan McBroom, Kimberly M. Burkhart, and Jacquelyn N. Pidruzny. The development of the game engagement questionnaire: A measure of engagement in video game-playing. *Journal of Experimental Social Psychology*, 45(4):624–634, 2009.

[BFH06] Hans H Bauer, Tomas Falk, and Maik Hammerschmidt. etransqual: A transaction process-based approach for capturing service quality in online shopping. *Journal of Business Research*, 59(7):866–875, 2006.

[Bis08] Pratik K Biswas. Towards an agent-oriented approach to conceptualization. *Applied Soft Computing*, 8(1):127–139, 2008.

[BK08] Gary Bradski and Adrian Kaehler. *Learning OpenCV: Computer vision with the OpenCV library*. 2008.

[BK15] Ayana Burkins and Regis Kopper. Wayfinding by auditory cues in virtual environments. In *2015 IEEE Virtual Reality (VR)*, pages 155–156, 2015.

[BKLMG13] Francesco Bellotti, Bill Kapralos, Kiju Lee, and Pablo Moreno-Ger. User assessment in serious games and technology-enhanced learning, Mar 2013.

[Bla96] Alan F Blackwell. Metacognitive theories of visual programming: what do we think we are doing? In *Proceedings 1996 IEEE symposium on visual languages*, pages 240–246. IEEE, 1996.

[BLKG05] Barry J Babin, Yong-Ki Lee, Eun-Ju Kim, and Mitch Griffin. Modeling consumer satisfaction and word-of-mouth: restaurant patronage in korea. *Journal of Services Marketing*, 2005.

[BLR92] Ronald J Brachman, Hector J Levesque, and Raymond Reiter. *Knowledge representation*. 1992.

[BM19] Sándor Bácsi and Gergely Mezei. Towards a classification to facilitate the design of domain-specific visual languages. *Acta Cybernetica*, 24(1):5–16, 2019.

[BMR07] Ahmed BinSubaih, Steve Maddock, and Daniela Romano. A survey of 'game' portability. *University of Sheffield, Tech. Rep. CS-07-05*, 2007.

[BRKD19] Evren Bozgeyikli, Andrew Raij, Srinivas Katkoori, and Rajiv Dubey. Locomotion in virtual reality for room scale tracked areas. *International Journal of Human-Computer Studies*, 122:38–49, 2019.

[Bru14] Liam Brummitt. Matter.js, a 2D rigid body physics engine for the web written in JavaScript. `https://brm.io/matter-js`, 2014. [Online; accessed June 30, 2022].

[BT10] Marius Bulearca and Daniel Tamarjan. Augmented reality: A sustainable marketing tool. *Global business and management research: An international journal*, 2(2):237–252, 2010.

[BV17] Leif P. Berg and Judy M. Vance. Industry use of virtual reality in product design and manufacturing: a survey. *Virtual Reality*, 21(1):1–17, Mar 2017.

[Cat11] Erin Catto. Box2D: a 2D physics engine for games. `https://box2d.org`, 2011. [Online; accessed June 30, 2022].

[CB20] David Checa and Andres Bustillo. A review of immersive virtual reality serious games to enhance learning and training. *Multimedia Tools and Applications*, 79(9):5501–5527, Mar 2020.

[CBJ+08] Carlos Carrascosa, Javier Bajo, Vicente Julián, Juan M Corchado, and Vicente Botti. Hybrid multi-agent architecture as a real-time problem-solving model. *Expert Systems with Applications*, 34(1):2–17, 2008.

[CC07] Woei-Kae Chen and Yu Chin Cheng. Teaching object-oriented programming laboratory with computer game programming. *IEEE Transactions on Education*, 50(3):197–203, 2007.

[CCE+10] Patrick Connolly, Cody Chambers, Evan Eagleson, Damian Matthews, and Tyler Rogers. Augmented reality effectiveness in advertising. In *65th Midyear Conference on Engineering Design Graphics Division of ASEE*, pages 3–6, 2010.

[CCPC01] Terry L Childers, Christopher L Carr, Joann Peck, and Stephen Carson. Hedonic and utilitarian motivations for online retail shopping behavior. *Journal of retailing*, 77(4):511–535, 2001.

[Cev20] Cevisama. Cevisama - International Fair for Ceramic Tiles and Bathroom Furnishings. `https://cevisama.feriavalencia.com`, 2020. [Online; accessed July 5, 2021].

[CFA+11] Julie Carmigniani, Borko Furht, Marco Anisetti, Paolo Ceravolo, Ernesto Damiani, and Misa Ivkovic. Augmented reality technologies, systems and applications. *Multimedia tools and applications*, 51(1):341–377, 2011.

[CGCC15] Eduardo H. Calvillo-Gámez, Paul Cairns, and Anna L. Cox. Assessing the core elements of the gaming experience. In Regina Bernhaupt, editor, *Game User Experience Evaluation*, pages 37–62. Cham, 2015.

[Cha05] Sheeson E Chang. Computer anxiety and perception of task complexity in learning programming-related skills. *Computers in Human Behavior*, 21(5):713–728, 2005.

[Cha16] Po-Yao Chao. Exploring students' computational practice, design and performance of problem-solving through a visual programming environment. *Computers & Education*, 95:202–215, 2016.

[CKW13] Hung-Lin Chi, Shih-Chung Kang, and Xiangyu Wang. Research trends and opportunities of augmented reality applications in architecture, engineering, and construction. *Automation in construction*, 33:116–122, 2013.

[CL16] C Chen and L Leung. Are you addicted to candy crush saga. *An exploratory study linking psychological factors to mobile*, 2016.

[CLCH17] Peng Chen, Xiaolin Liu, Wei Cheng, and Ronghuai Huang. A review of using augmented reality in education from 2011 to 2016. *Innovations in smart learning*, pages 13–18, 2017.

[CMLRR20] Miguel Chover, Carlos Marín-Lora, Cristina Rebollo, and Inmaculada Remolar. A game engine designed to simplify 2d video game development. *Multimedia Tools and Applications*, 79(17):12307–12328, 2020.

[CNdSR16] Cristiano Carvalheiro, Rui Nóbrega, Hugo da Silva, and Rui Rodrigues. User redirection and direct haptics in virtual environments. In *Proceedings of the 24th ACM International Conference on Multimedia*, MM '16, pages 1146–1155, New York, NY, USA, 2016.

[CNS05] David Callele, Eric Neufeld, and Kevin Schneider. Requirements engineering and the creative process in the video game industry. In *13th IEEE International Conference on Requirements Engineering (RE'05)*, pages 240–250. IEEE, 2005.

[COB+18] Inrak Choi, Eyal Ofek, Hrvoje Benko, Mike Sinclair, and Christian Holz. Claw: A multifunctional handheld haptic controller for grasping, touching, and triggering in virtual reality. In *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems*, CHI '18, pages 1–13, New York, NY, USA, 2018.

[COEGD+19] Edmanuel Cruz, Sergio Orts-Escolano, Francisco Gomez-Donoso, Carlos Rizo, Jose Carlos Rangel, Higinio Mora, and Miguel Cazorla. An augmented reality application for improving shopping experience in large retail stores. *Virtual Reality*, 23(3):281–291, 2019.

[Cor15] Jose David Cuartas Correa. *Digitopolis II: Creación de videojuegos con GDevelop*. 2015.

[Cox14] Joe Cox. What makes a blockbuster video game? an empirical analysis of us sales data. *Managerial and Decision Economics*, 35(3):189–198, 2014.

[Dac17] Scott G Dacko. Enabling smart retail settings via mobile augmented reality shopping apps. *Technological Forecasting and Social Change*, 124:243–256, 2017.

[Dai10] Abdellah Daissaoui. Applying the MDA approach for the automatic generation of an MVC2 web application. In *2010 Fourth International Conference on Research Challenges in Information Science (RCIS)*, pages 681–688. IEEE, 2010.

[Dav89] Fred D Davis. Perceived usefulness, perceived ease of use, and user acceptance of information technology. *MIS quarterly*, pages 319–340, 1989.

[DeL00] Mark DeLoura. *Game programming gems*. 2000.

[DKJ18] Ali Dorri, Salil S Kanhere, and Raja Jurdak. Multi-agent systems: A survey. *Ieee Access*, 6:28573–28593, 2018.

[DNC16] Alena Denisova, A. Imran Nordin, and Paul Cairns. The convergence of player experience questionnaires. In *Proceedings of the 2016 Annual Symposium on Computer-Human Interaction in Play*, CHI PLAY '16, pages 33–37, New York, NY, USA, 2016.

[Doh03] Michael Doherty. A software architecture for games. *University of the Pacific Department of Computer Science Research and Project Journal (RAPJ)*, 1(1), 2003.

[dSCMZ20] Luís Fernando de Souza Cardoso, Flávia Cristina Martins Queiroz Mariano, and Ezequiel Roberto Zorzal. A survey of industrial augmented reality. *Computers & Industrial Engineering*, 139:106159, 2020.

[DV04] Fred D Davis and Viswanath Venkatesh. Toward preprototype user acceptance testing of new information systems: implications for software project management. *IEEE Transactions on Engineering management*, 51(1):31–46, 2004.

[DVRL12] Denis V Dorozhkin, Judy M Vance, Gordon D Rehn, and Marco Lemessi. Coupling of interactive manufacturing operations simulation and immersive virtual reality. *Virtual Reality*, 16(1):15–23, 2012.

[DWvDH09] Frank Dignum, Joost Westra, Willem A van Doesburg, and Maaike Harbers. Games and agents: Designing intelligent gameplay. *International Journal of Computer Games Technology*, 2009, 2009.

[DX12] Sonal Dekhane and Xin Xu. Engaging students in computing using gamesalad: a pilot study. *Journal of Computing Sciences in Colleges*, 28(2):117–123, 2012.

[ELL+13] Marc Ebner, John Levine, Simon M Lucas, Tom Schaul, Tommy Thompson, and Julian Togelius. Towards a video game description language. *Dagstuhl Publishing*, 2013.

[Epi22] EpicGames. Unreal Engine 4: The world's most open and advanced real-time 3D creation tool. `https://www.unrealengine.com`, 2022. [Online; accessed June 30, 2022].

[ESA17] ESA Entertainment Software Association. Essential Facts About the Computer and Video Game Industry. `https://www.theesa.com/resource/2017-essential-facts-about-the-computer-and-video-game-industry`, 2017. [Online; accessed December 17, 2021].

[FB08] Hilmar Finnsson and Yngvi Björnsson. Simulation-based approach to general game playing. In *AAAI*, volume 8, pages 259–264, 2008.

[FdSB16] Mauricio A Frigo, EC da Silva, and Gustavo F Barbosa. Augmented reality in aerospace manufacturing: A review. *Journal of Industrial and Intelligent Information*, 4(2), 2016.

[FHMV04] Ronald Fagin, Joseph Y Halpern, Yoram Moses, and Moshe Vardi. *Reasoning about knowledge*. 2004.

[FLFCBNVM18] Paula Fraga-Lamas, Tiago M Fernandez-Carames, Oscar Blanco-Novoa, and Miguel A Vilar-Montesinos. A review on industrial augmented reality systems for the industry 4.0 shipyard. *Ieee Access*, 6:13358–13375, 2018.

[Flo19] FlowlabIO. Flowlab. `https://flowlab.io`, 2019. [Online; accessed January 8, 2019].

[Fol07] Eelke Folmer. Component based game development–a solution to escalating costs and expanding deadlines? In *International symposium on component-based software engineering*, pages 66–73. Springer, 2007.

[Fri37] Milton Friedman. The use of ranks to avoid the assumption of normality implicit in the analysis of variance. *Journal of the Amer Stat Assoc*, 32(200):675–701, 1937.

[FS06] André WB Furtado and André LM Santos. Using domain-specific modeling towards computer games development industrialization. In *The 6th OOPSLA workshop on domain-specific modeling (DSM06)*, 2006.

[FSRdA11] Andre WB Furtado, Andre LM Santos, Geber L Ramalho, and Eduardo Santana de Almeida. Improving digital game development with software product lines. *IEEE software*, 28(5):30–37, 2011.

[FW99] Jacques Ferber and Gerhard Weiss. *Multi-agent systems: an introduction to distributed artificial intelligence*, volume 1. 1999.

[Gal08] R Galindo. *Presses, molds and compaction in the manufacture of ceramic tiles*. 2008.

[Gam21] Gamesonomy. Gamesonomy - make games in the cloud and play in your phone. `https://gamesonomy.com/`, 2021. [Online; accessed July 30, 2021].

[GBF+16] Andrés Ayala García, Israel Galván Bobadilla, Gustavo Arroyo Figueroa, Miguel Pérez Ramírez, and Javier Muñoz Román. Virtual reality training system for maintenance and operation of high-voltage overhead power lines. *Virtual Reality*, 20(1):27–40, 2016.

[GDe22] GDevelop. GDevelop - Free and Easy Game-Making App. `https://gdevelop.io/`, 2022. [Online; accessed June 30, 2022].

[Gel03] James Geller. Knowledge representation: Logical, philosophical, and computational foundations, brooks/cole, 2000, 512pp. *Minds and Machines*, 13(3):441–444, 2003.

[Geo75] Robert A. Georges. *Western Folklore*, 34(2):155–158, 1975.

[GFZI12] Bin Guo, Ryota Fujimura, Daqing Zhang, and Michita Imai. Design-in-play: improving the variability of indoor pervasive games. *Multimedia Tools and Applications*, 59(1):259–277, 2012.

[Gil92] Valerie J Gilchrist. Key informant interviews. *Sage Publications, Inc*, 1992.

[GJMSMCMJ14] Sergio Garrido-Jurado, Rafael Muñoz-Salinas, Francisco José Madrid-Cuevas, and Manuel Jesús Marín-Jiménez. Automatic generation and detection of highly reliable fiducial markers under occlusion. *Pattern Recognition*, 47(6):2280–2292, 2014.

[GKN11] Kathrin M. Gerling, Matthias Klauser, and Joerg Niesenhaus. Measuring the impact of game controllers on player experience in fps games. In *Proceedings of the 15th International Academic MindTrek Conference: Envisioning Future Media Environments*, MindTrek '11, pages 83 - 86, New York, NY, USA, 2011.

[GLP05] Michael Genesereth, Nathaniel Love, and Barney Pell. General game playing: Overview of the AAAI competition. *AI magazine*, 26(2):62–62, 2005.

[GN12] Michael Genesereth and Nils J Nilsson. *Logical foundations of artificial intelligence*. 2012.

[Gol99] Reginald G Golledge. Human wayfinding and cognitive maps. volume 5, page 45, 1999.

[Goo13] Mat Groves GoodBoyDigital. PixiJS — The HTML5 Creation Engine. `https://pixijs.com/`, 2013. [Online; accessed June 30, 2022].

[Goo19] Google. Google Material Design. `https://design.google`, 2019. [Online; accessed January 8, 2019].

[Goo21] Google. ARCore - Google Developers. `https://developers.google.com/ar`, 2021. [Online; accessed July 5, 2021].

[GQC+07a] Alejandro Garcés, Ricardo Quirós, Miguel Chover, Joaquin Huerta, and Emilio Camahort. A development methodology for moderately open multi-agent systems. In *Proceedings of the 25th conference on IASTED International Multi-Conference: Software Engineering, SE*, volume 7, pages 37–42, 2007.

[GQC+07b] Alejandro Garcés, Ricardo Quirós, Miguel Chover, Joaquín Huerta, and Emilio Camahort. E-commerce transaction modeling using moderately open multi-agent systems. In *ICEIS (4)*, pages 167–172, 2007.

[GQCC06] Alejandro Garcés, Ricardo Quirós, Miguel Chover, and Emilio Camahort. Implementing moderately open agent-based systems. In *IADIS International Conference WWW/Internet 2006*, pages 360–369, 2006.

[GQCC10] Alejandro Garcés, Ricardo Quirós, Miguel Chover, and Emilio Camahort. Implementing virtual agents: a haba-based approach. *The International journal of Multimedia & Its Applications (IJMA) Vol*, 2, 2010.

[Gre18] Jason Gregory. *Game engine architecture*. 2018.

[GSF13] McClure Grant, Virwaney Sandeep, and Lin Fuhua. Integrating multiagent systems into virtual worlds. In *3rd International Conference on Multimedia Technology (ICMT-13)*, pages 574–581. Atlantis Press, 2013.

[GYSE20] Sebastian Gottschalk, Enes Yigitbas, Eugen Schmidt, and Gregor Engels. Model-based product configuration in augmented reality applications. In *International Conference on Human-Centred Software Engineering*, pages 84–104. Springer, 2020.

[HdRC+17] Tim Hilken, Ko de Ruyter, Mathew Chylinski, Dominik Mahr, and Debbie I Keeling. Augmenting the eye of the beholder: exploring the strategic potential of augmented reality to enhance online service experiences. *Journal of the Academy of Marketing Science*, 45(6):884–905, 2017.

[HK17] Seunghun Han and Jinmo Kim. A study on immersion of hand interaction for mobile platform virtual reality contents. *Symmetry*, 9(2), 2017.

[HLZ04] Robin Hunicke, Marc LeBlanc, and Robert Zubek. A formal approach to game design and game research. *GDC. San Jose*, 2004.

[HORB08] Kristin Hanks, William Odom, David Roedl, and Eli Blevis. Sustainable millennials: attitudes towards sustainability and the material effects of interactive technologies. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 333–342, 2008.

[HŠM+19] Nikola Horvat, Stanko Škec, Tomislav Martinec, Fanika Lukačević, and Marija Majda Perišić. Comparing virtual reality and desktop interface for reviewing 3d cad models. volume 1, page 1923–1932, 2019.

[HWV+15] Tilo Hartmann, Werner Wirth, Peter Vorderer, Christoph Klimmt, Holger Schramm, and Saskia Böcking. Spatial presence theory: State of the art and challenges ahead. *Immersed in media*, pages 115–135, 2015.

[HZQ+19] Zhijiong Huang, Yu Zhang, Kathryn C. Quigley, Ramya Sankar, Clemence Wormser, Xinxin Mo, and Allen Y. Yang. Accessibility of virtual reality locomotion modalities to adults and minors. *arXiv*, 2019.

[IdP13] W.A. IJsselsteijn, Y.A.W. de Kort, and K. Poels. The game experience questionnaire. *Technische Universiteit Eindhoven*, 2013.

[JBT+18] Arthur Juliani, Vincent-Pierre Berges, Ervin Teng, Andrew Cohen, Jonathan Harper, Chris Elion, Chris Goy, Yuan Gao, Hunter Henry, Marwan Mattar, et al. Unity: A general platform for intelligent agents. *arXiv preprint arXiv:1809.02627*, 2018.

[JER18] Jérme Jetter, Jörgen Eimecke, and Alexandra Rese. Augmented reality tools for industrial applications: What are potential key performance indicators and who benefits? *Computers in Human Behavior*, 87:18–33, 2018.

[JFP10] Pauline Jepp, Manuel Fradinho, and João Madeiras Pereira. An agent framework for a modular serious game. In *2010 Second International Conference on Games and Virtual Worlds for Serious Applications*, pages 19–26. IEEE, 2010.

[JHMWA18] M. P. Jacob Habgood, David Moore, David Wilson, and Sergio Alapont. Rapid, continuous movement between nodes as an accessible virtual reality locomotion technique. In *2018 IEEE Conference on Virtual Reality and 3D User Interfaces (VR)*, pages 371–378, 2018.

[JKK+20] Kisung Jeong, Jinmo Kim, Mingyu Kim, Jiwon Lee, and Chanhun Kim. Asymmetric interface: User interface of asymmetric virtual reality for new presence and experience. *Symmetry*, 12(1), 2020.

[JOW16] Karl G Jöreskog, Ulf H Olsson, and Fan Y Wallentin. *Multivariate analysis with LISREL*. 2016.

[Juu10] Jesper Juul. The game, the player, the world: Looking for a heart of gameness. *Plurais Revista Multidisciplinar*, 1(2), 2010.

[Juu11] Jesper Juul. *Half-real: Video games between real rules and fictional worlds*. 2011.

[Kar88] Kevin Karplus. *Using if-then-else DAGs for multi-level logic minimization*. 1988.

[KAT+16] Varma Kamadi, Appa Rao Allam, Sita Mahalakshmi Thummala, et al. A computational intelligence technique for the effective diagnosis of diabetic patients using principal component analysis (pca) and modified fuzzy sliq decision tree approach. *Applied Soft Computing*, 49:137–145, 2016.

[Ken19] KenneyArt. Abstract platformer. `https://www.kenney.nl/assets/abstract-platformer`, 2019. [Online; accessed July 1, 2019].

[KHA97] James D Kiper, Elizabeth Howard, and Chuck Ames. Criteria for evaluation of visual programming languages. *Journal of Visual Languages & Computing*, 8(2):175–192, 1997.

[KHF+19] Julian Kreimeier, Sebastian Hammer, Daniel Friedmann, Pascal Karg, Clemens Bühner, Lukas Bankel, and Timo Götzelmann. Evaluation of different types of haptic feedback influencing the task-based presence and performance in virtual reality. In *Proceedings of the 12th ACM International Conference on Pervasive Technologies Related to Assistive Environments*, PETRA '19, pages 289–298, New York, NY, USA, 2019.

[Kim16] Jinmo Kim. Modeling and optimization of a tree based on virtual reality for immersive virtual landscape generation. *Symmetry*, 8(9), 2016.

[Kir04] Andrew Kirmse. *Game Programming Gems 4 (Game Programming Gems Series)*. 2004.

[Kle03] Lisa R Klein. Creating virtual product experiences: The role of telepresence. *Journal of interactive Marketing*, 17(1):41–55, 2003.

[KLM14] Theodora Koulouri, Stanislao Lauria, and Robert D Macredie. Teaching introductory programming: A quantitative evaluation of different approaches. *ACM Transactions on Computing Education (TOCE)*, 14(4):1–28, 2014.

[Kos13] Raph Koster. *Theory of fun for game design*. 2013.

[KS81] Konami-SEGA. Frogger. `https://en.wikipedia.org/wiki/Frogger`, 1981. [Online; accessed June 30, 2022].

[LB03] Sus Lundgren and Staffan Bjork. Game mechanics: Describing computer-augmented games in terms of interaction. In *Proceedings of TIDSE*, volume 3, 2003.

[LBM18] Effie L.-C. Law, Florian Brühlmann, and Elisa D. Mekler. Systematic review and validation of the game experience questionnaire (geq) - implications for citation and reporting practice. In *Proceedings of the 2018 Annual Symposium on Computer-Human Interaction in Play*, CHI PLAY '18, pages 257-270, New York, NY, USA, 2018.

[LCSP12] Pedro Latorre Carmona, José Martínez Sotoca, and Filiberto Pla. Filter-type variable selection based on information measures for regression tasks. *Entropy*, 14(2):323–343, 2012.

[LDB+05] Hairong Li, Terry Daugherty, Frank Biocca, MR Stafford, and RJ Faber. Impact of 3d advertising on product knowledge, brand attitude and purchase intention. In *Advertising, Promotion and New Media*, page 149. 2005.

[Lee12] Kangdon Lee. Augmented reality in education and training. *TechTrends*, 56(2):13–21, 2012.

[LFH17] Jonathan Lazar, Jinjuan Heidi Feng, and Harry Hochheiser. *Research methods in human-computer interaction*. 2017.

[LHH+08] Nathaniel Love, Timothy Hinrichs, David Haley, Eric Schkufza, and Michael Genesereth. General game playing: Game description language specification. *Stanford Logic Group Computer Science Department Stanford University ...*, 2008.

[LJ02] Michael Lewis and Jeffrey Jacobson. Game engines. *Communications of the ACM*, 45(1):27, 2002.

[LLW+14] Jiangjiang Liu, Cheng-Hsien Lin, Joshua Wilson, David Hemmenway, Ethan Hasson, Zebulun Barnett, and Yingbo Xu. Making games a" snap" with stencyl: a summer computing workshop for k-12 teachers. In *Proceedings of the 45th ACM technical symposium on Computer science education*, pages 169–174, 2014.

[LNO17] Wenkai Li, AYC Nee, and SK Ong. A state-of-the-art review of augmented reality in engineering analysis and simulation. *Multimodal Technologies and Interaction*, 1(3):17, 2017.

[LS07] Yuzhu Lu and Shana Smith. Augmented reality e-commerce assistant system: trying while shopping. In *International Conference on Human-Computer Interaction*, pages 643–652. Springer, 2007.

[LSBF17] Daniele Leonardis, Massimiliano Solazzi, Ilaria Bortone, and Antonio Frisoli. A 3-rsr haptic wearable device for rendering fingertip contact forces. *IEEE Transactions on Haptics*, 10(3):305–316, 2017.

[LSGF+19] Jaeyeon Lee, Mike Sinclair, Mar Gonzalez-Franco, Eyal Ofek, and Christian Holz. Torc: A virtual reality controller for in-hand high-dexterity finger interaction. In *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems*, CHI '19, pages 1-13, New York, NY, USA, 2019.

[LTC21] Priscilla Lo, David Thue, and Elin Carstensdottir. What is a game mechanic? In *International Conference on Entertainment Computing*, pages 336–347. Springer, 2021.

[ME11] Frans Mäyrä and Laura Ermi. Fundamental components of the gameplay experience. *DIGAREC Series*, (6):88 – 115, 2011.

[Mil10] I Millington. How to build a robust commercial-grade physics engine for your game. *Game Physics Engine Development, 2nd Edition, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA*, 2010.

[Mil19] Ian Millington. *AI for Games*. 2019.

[ML20] Carlos Marín-Lora. Games developed as demonstrators in the use cases. `https://sites.google.com/uji.es/multiagent-gameengine`, 2020. [Online; accessed July 30, 2019].

[MLCS19] Carlos Marín-Lora, Miguel Chover, and José M Sotoca. Prototyping a game engine architecture as a multi-agent system. In *27th International Conference in Central Europe on Computer Graphics, Visualization and Computer Vision (WSCG 2019)*, 2019.

[MLCSG20] Carlos Marín-Lora, Miguel Chover, José M Sotoca, and Luis A García. A game engine to make games as multi-agent systems. *Advances in Engineering Software*, 140:102732, 2020.

[MLV14] Naresh K Malhotra, Evandro Lopes, and Ricardo Teixeira Veiga. Structural equation modeling with lisrel: An initial vision. *Brazilian Journal of Marketing*, 13(2), 2014.

[Mol06] Jules Moloney. Augmented reality visualisation of the built environment to support design decision making. In *Tenth International Conference on Information Visualisation (IV'06)*, pages 687–692. IEEE, 2006.

[MP17] Lakmal Meegahapola and Indika Perera. Enhanced in-store shopping experience through smart phone based mixed reality application. In *2017 Seventeenth International Conference on Advances in ICT for Emerging Regions (ICTer)*, pages 1–8. IEEE, 2017.

[MR02] Iain Milne and Glenn Rowe. Difficulties in learning and teaching programming—views of students and tutors. *Education and Information technologies*, 7(1):55–66, 2002.

[MRR+10] John Maloney, Mitchel Resnick, Natalie Rusk, Brian Silverman, and Evelyn Eastmond. The scratch programming language and environment. *ACM Transactions on Computing Education (TOCE)*, 10(4):1–15, 2010.

[MSG+15] Erin Martel, Feng Su, Jesse Gerroir, Ahmed Hassan, Audrey Girouard, and Kasia Muldner. Diving head-first into virtual reality: Evaluating hmd control schemes for vr games. In *Proceedings of the 10th International Conference on the Foundations of Digital Games (FDG 2015)*, Pacific Grove, CA, USA, 2015.

[MSK15] Farouk Messaoudi, Gwendal Simon, and Adlen Ksentini. Dissecting games engines: The case of unity3d. In *2015 international workshop on network and systems support for games (NetGames)*, pages 1–6. IEEE, 2015.

[MW12] Michelle Menard and Bryan Wagstaff. *Game development with Unity*. 2012.

[NDC14] A. Imran. Nordin, Alena Denisova, and Paul Cairns. Too many questionnaires: Measuring player experience whilst playing digital games. September 2014.

[Nic05] Esposito Nicolas. A short and simple definition of what a videogame is. *University of Technology of Compiègne*, 2005.

[Nys14] Robert Nystrom. *Game programming patterns*. 2014.

[OD98] Michael Occello and Yves Demazeau. Modelling decision making systems using agents for cooperation in a real time constraints. In *3rd IFAC Symposium on Intelligent Autonomous Vehicles*, volume 1, pages 51–56, 1998.

[OKD+15] Ibrahim Ouahbi, Fatiha Kaddari, Hassane Darhmaoui, Abdelrhani Elachqar, and Soufiane Lahmine. Learning basic programming concepts by creating games with scratch programming environment. *Procedia-Social and Behavioral Sciences*, 191:1479–1482, 2015.

[OS06] Reza Olfati-Saber. Flocking for multi-agent dynamic systems: Algorithms and theory. *IEEE Transactions on automatic control*, 51(3):401–420, 2006.

[OSFM07] Reza Olfati-Saber, J Alex Fax, and Richard M Murray. Consensus and cooperation in networked multi-agent systems. *Proceedings of the IEEE*, 95(1):215–233, 2007.

[Pae14] Volker Paelke. Augmented reality in the smart factory: Supporting workers in an industry 4.0. environment. In *Proceedings of the 2014 IEEE emerging technology and factory automation (ETFA)*, pages 1–4. IEEE, 2014.

[PB13] Luc Pons and Carole Bernon. A multi-agent system for autonomous control of game parameters. In *2013 IEEE International Conference on Systems, Man, and Cybernetics*, pages 583–588. IEEE, 2013.

[PCPM13] Domenico Prattichizzo, Francesco Chinello, Claudio Pacchierotti, and Monica Malvezzi. Towards wearability in fingertip haptics: A 3-dof wearable device for cutaneous force feedback. *IEEE Transactions on Haptics*, 6(4):506–516, 2013.

[PGC+06] Kris Powers, Paul Gross, Steve Cooper, Myles McNally, Kenneth J Goldman, Viera Proulx, and Martin Carlisle. Tools for teaching introductory programming: what works? In *Proceedings of the 37th SIGCSE technical symposium on Computer science education*, pages 560–561, 2006.

[PKC16] Mikki H. Phan, Joseph R. Keebler, and Barbara S. Chaparro. The development and validation of the game user experience satisfaction scale (guess). *Human Factors*, 58(8):1217–1247, 2016. PMID: 27647156.

[PLST+16] Diego Perez-Liebana, Spyridon Samothrakis, Julian Togelius, Tom Schaul, and Simon M Lucas. General video game ai: Competition, challenges and opportunities. In *Thirtieth AAAI conference on artificial intelligence*, 2016.

[PLX16] Jeffrey R Parker, Donald R Lehmann, and Yi Xie. Decision comfort. *Journal of Consumer Research*, 43(1):113–133, 2016.

[PM14] Ingrid Poncin and Mohamed Slim Ben Mimoun. The impact of "e-atmospherics" on physical stores. *Journal of Retailing and Consumer Services*, 21(5):851–859, 2014.

[Pos07] Stefan Poslad. Specifying protocols for multi-agent systems interaction. *ACM Transactions on Autonomous and Adaptive Systems (TAAS)*, 2(4):15–es, 2007.

[PP19] Federica Pallavicini and Alessandro Pepe. Comparing player experience in video games played in virtual reality or on desktop displays: Immersion, flow, and positive emotions. In *Extended Abstracts of the Annual Symposium on Computer-Human Interaction in Play Companion Extended Abstracts*, CHI PLAY '19 Extended Abstracts, pages 195-210, New York, NY, USA, 2019.

[PPM19] Federica Pallavicini, Alessandro Pepe, and Maria Eleonora Minissi. Gaming in virtual reality: What changes in terms of usability, emotional response and sense of presence compared to non-immersive video games? *Simulation & Gaming*, 50(2):136–159, 2019.

[PS85] Peter F Patel-Schneider. A decidable first-order logic for knowledge representation. In *IJCAI*, volume 9, pages 455–458, 1985.

[PW02] Simon Parsons and Michael Wooldridge. Game theory and decision theory in multi-agent systems. *Autonomous Agents and Multi-Agent Systems*, 5(3):243–254, 2002.

[RA03] Andrew Rollings and Ernest Adams. *Andrew Rollings and Ernest Adams on game design*. 2003.

[RBF21] Mehrdad Rostami, Kamal Berahmand, and Saman Forouzandeh. A novel community detection based genetic algorithm for feature selection. *Journal of Big Data*, 8(1):1–27, 2021.

[RBNF21] Mehrdad Rostami, Kamal Berahmand, Elahe Nasiri, and Saman Forouzandeh. Review of swarm intelligence-based feature selection methods. *Engineering Applications of Artificial Intelligence*, 100:104210, 2021.

[RC08] Emanuel Montero Reyno and José Á Carsí Cubel. Model driven game development: 2d platform game prototyping. In *GAMEON*, pages 5–7. Citeseer, 2008.

[RCRG06] Francisco Ramos, Miguel Chover, Oscar Ripolles, and Carlos Granell. Continuous level of detail on graphics hardware. In *International Conference on Discrete Geometry for Computer Imagery*, pages 460–469. Springer, 2006.

[RFBS20] Mehrdad Rostami, Saman Forouzandeh, Kamal Berahmand, and Mina Soltani. Integration of multi-objective pso based feature selection and node centrality for medical datasets. *Genomics*, 112(6):4370–4384, 2020.

[RFH19] Philipp A Rauschnabel, Reto Felix, and Chris Hinsch. Augmented reality marketing: How mobile ar-apps can improve brands through inspiration. *Journal of Retailing and Consumer Services*, 49:43–53, 2019.

[RGR+15] Inmaculada Remolar, Alejandro Garcés, Cristina Rebollo, Miguel Chover, Ricardo Quirós, and Jesús Gumbau. Developing a virtual trade fair using an agent-oriented approach. *Multimedia Tools and Applications*, 74(13):4561–4582, 2015.

[RMLRC18] Cristina Rebollo, Carlos Marín-Lora, Inmaculada Remolar, and Miguel Chover. Gamesonomy vs scratch: two different ways to introduce programming. In *15th International Conference On Cognition And Exploratory Learning In The Digital Age (CELDA 2018). Ed. IADIS Press*, 2018.

[RPB17] Tero Reunanen, Marcus Penttinen, and Arndt Borgmeier. "wow-factors" for boosting business. In Jussi Ilari Kantola, Tibor Barath, Salman Nazir, and Terence Andre, editors, *Advances in Human Factors, Business Management, Training and Education*, pages 589–600. Cham, 2017.

[RPG19] RPGMaker. RPG Maker - Make Your Own Game with RPG Maker. `http://www.rpgmakerweb.com`, 2019. [Online; accessed July 12, 2019].

[RRD12] Krishnendu Roy, William C Rousse, and David B DeMeritt. Comparing the mobile novice programming environments: App inventor for android vs. gamesalad. In *2012 Frontiers in education conference proceedings*, pages 1–6. IEEE, 2012.

[RRPCC12] Oscar Ripolles, Francisco Ramos, Anna Puig-Centelles, and Miguel Chover. Real-time tessellation of terrain on graphics hardware. *Computers & geosciences*, 41:147–155, 2012.

[RRR03] Anthony Robins, Janet Rountree, and Nathan Rountree. Learning and teaching programming: A review and discussion. *Computer science education*, 13(2):137–172, 2003.

[RS94] Stuart J Russell and Devika Subramanian. Provably bounded-optimal agents. *Journal of Artificial Intelligence Research*, 2:575–609, 1994.

[Sch09] Thomas W Schubert. A new conception of spatial presence: Once again, with feeling. *Communication Theory*, 19(2):161–187, 2009.

[SCT03] Carla TLL Silva, Jaelson Castro, and Patricia Azevedo Tedesco. Requirements for multi-agent systems. *WER*, 2003:198–212, 2003.

[SIB11] George Sacerdotianu, Sorin Ilie, and Costin Badica. Software framework for agent-based games and simulations. In *2011 13th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing*, pages 381–388. IEEE, 2011.

[Sic08] Miguel Sicart. Defining game mechanics. *Game studies*, 8(2):1–14, 2008.

[SK14] Philipp Spreer and Katrin Kallweit. Augmented reality in retail: assessing the acceptance and potential for multimedia product presentation at the pos. *Transactions on Marketing Research*, 1(1):20–35, 2014.

[SL17] Lee Stemkoski and Evan Leider. *Game Development with Construct 2: From Design to Realization*. 2017.

[SNM16] Carl Schissler, Aaron Nicholls, and Ravish Mehra. Efficient hrtf-based spatial audio for area and volumetric sources. *IEEE Transactions on Visualization and Computer Graphics*, 22(4):1356–1366, 2016.

[ST09] Stephan Schiffel and Michael Thielscher. A multiagent semantics for the game description language. In *International conference on agents and artificial intelligence*, pages 44–55. Springer, 2009.

[Ste19] Stencyl. Stencyl: Make iPhone, iPad, Android Flash Games without code. `https://www.stencyl.com`, 2019. [Online; accessed June 30, 2019].

[Sut65] Ivan E. Sutherland. The ultimate display. In *Proceedings of the Congress of the Internation Federation of Information Processing (IFIP)*, volume volume 2, pages 506–508, 1965.

[SW14] David Sharek and Eric Wiebe. Measuring video game engagement through the cognitive and affective dimensions. *Simulation & Gaming*, 45(4-5):569–592, 2014.

[Tan08] Eduard Sioe-Hao Tan. Entertainment is emotion: The functional architecture of the entertainment experience. *Media Psychology*, 11(1):28–51, 2008.

[TAZF17] Sam Tregillus, Majed Al Zayer, and Eelke Folmer. Handsfree omnidirectional vr navigation using head tilt. In *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems*, CHI '17, pages 4063 - 4068, New York, NY, USA, 2017.

[Thi10] Michael Thielscher. A general game description language for incomplete information games. In *Twenty-Fourth AAAI Conference on Artificial Intelligence*, 2010.

[Thi11] Michael Thielscher. The general game playing description language is universal. In *Twenty-Second International Joint Conference on Artificial Intelligence*, 2011.

[TLS+15] Chek Tien Tan, Tuck Wah Leong, Songjia Shen, Christopher Dubravs, and Chen Si. Exploring gameplay experiences on the oculus rift. In *Proceedings of the 2015 Annual Symposium on Computer-Human Interaction in Play*, CHI PLAY '15, pages 253 – 263, New York, NY, USA, 2015.

[Tuf03] Edward R Tufte. *The cognitive style of PowerPoint*, volume 2006. 2003.

[TW04] Seth Tisue and Uri Wilensky. Netlogo: A simple environment for modeling complexity. In *International conference on complex systems*, volume 21, pages 16–21. Boston, MA, 2004.

[TZ05] Katie Salen Tekinbas and Eric Zimmerman. *The game design reader: A rules of play anthology*. 2005.

[Uni19] UnityTechnologies. Unity Tower Bridge Defense tutorial. `https://unity3d.com/es/learn/tutorials/topics/2d-game-creation/2d-game-development-walkthrough`, 2019. [Online; accessed July 1, 2019].

[Uni21] UnityTechnologies. Unity 3D Game Engine 2021. `https://unity3d.com/es/beta/2021.1b`, 2021. [Online; accessed 5-Jul-2021].

[Uni22] UnityTechnologies. Unity: The world's leading platform for real-time content creation. `https://unity.com`, 2022. [Online; accessed June 30, 2022].

[VAKAM20] Jan-Niklas Voigt-Antons, Tanja Kojic, Danish Ali, and Sebastian Möller. Influence of hand tracking as a way of interaction in virtual reality on user experience. In *2020 Twelfth International Conference on Quality of Multimedia Experience (QoMEX)*, pages 1–4, 2020.

[Val15] Nicola Valcasara. *Unreal engine game development blueprints*. 2015.

[VdBLM08] Jur Van den Berg, Ming Lin, and Dinesh Manocha. Reciprocal velocity obstacles for real-time multi-agent navigation. In *2008 IEEE international conference on robotics and automation*, pages 1928–1935. Ieee, 2008.

[VdS15] Rex Van der Spuy. *Learn Pixi. js*. 2015.

[VKBS13] Khrystyna Vasylevska, Hannes Kaufmann, Mark Bolas, and Evan A. Suma. Flexible spaces: Dynamic layout generation for infinite walking in virtual environments. In *2013 IEEE Symposium on 3D User Interfaces (3DUI)*, pages 39–42, 2013.

[VKP10] Rick Van Krevelen and Ronald Poelman. A survey of augmented reality technologies, applications and limitations. *International journal of virtual reality*, 9(2):1–20, 2010.

[VMB+13] Edward Rolando Núñez Valdez, Óscar Sanjuán Martínez, Begoña Cristina Pelayo García Bustelo, Juan Manuel Cueva Lovelle, and Guillermo Infante Hernandez. Gade4all: developing multi-platform videogames based on domain specific languages and model driven engineering. *IJIMAI*, 2(2):33–42, 2013.

[Vuf21] Vuforia. Vuforia Engine. `https://developer.vuforia.com/`, 2021. [Online; accessed 5-Jul-2021].

[VWG+04] Peter Vorderer, Werner Wirth, Feliz R Gouveia, Frank Biocca, Timo Saari, Futz Jäncke, Saskia Böcking, Holger Schramm, Andre Gysbers, Tilo Hartmann, et al. Mec spatial presence questionnaire (mec-spq): Short documentation and instructions for application. *Report to the European community, project presence: MEC (IST-2001-37661)*, 3:5–3, 2004.

[WFM+96] Anthony Webster, Steven Feiner, Blair MacIntyre, William Massie, and Theodore Krueger. Augmented reality in architectural construction, inspection and renovation. In *Proc. ASCE Third Congress on Computing in Civil Engineering*, volume 1, page 996, 1996.

[WFR+17] Marcel Walch, Julian Frommel, Katja Rogers, Felix Schüssel, Philipp Hock, David Dobbelstein, and Michael Weber. Evaluating vr driving simulation from a player experience perspective. In *Proceedings of the 2017 CHI Conference Extended Abstracts on Human Factors in Computing Systems*, CHI EA '17, page 2982 2989, New York, NY, USA, 2017.

[Wil99] Uri Wilensky. NetLogo. `http://ccl.northwestern.edu/netlogo/`, 1999. [Online; accessed January 30, 2022].

[Wil01] Uri Wilensky. NetLogo Tetris model. `http://ccl.northwestern.edu/netlogo/models/Tetris`, 2001. [Online; accessed January 30, 2022].

[Wil02] Dmitri Williams. Structure and competition in the us home video game industry. *International Journal on Media Management*, 4(1):41–54, 2002.

[Win06] Jeannette M Wing. Computational thinking. *Communications of the ACM*, 49(3):33–35, 2006.

[Win08] Jeannette M Wing. Computational thinking and thinking about computing. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 366(1881):3717–3725, 2008.

[WJ95] Michael Wooldridge and Nicholas R Jennings. Intelligent agents: Theory and practice. *The knowledge engineering review*, 10(2):115–152, 1995.

[WJK08] Chenggang Wang, Saket Joshi, and Roni Khardon. First order decision diagrams for relational mdps. *Journal of Artificial Intelligence Research*, 31:431–472, 2008.

[Woo09] Michael Wooldridge. *An introduction to multiagent systems*. 2009.

[WWH+08] David Weibel, Bartholomäus Wissmath, Stephan Habegger, Yves Steiner, and Rudolf Groner. Playing online games against computer- vs. human-controlled opponents: Effects on presence, flow, and enjoyment. *Computers in Human Behavior*, 24(5):2274–2291, 2008. Including the Special Issue: Internet Empowerment.

[YCS17] Mark Yi-Cheon Yim, Shu-Chuan Chu, and Paul L Sauer. Is augmented reality technology an effective tool for e-commerce? an interactivity and vividness perspective. *Journal of Interactive Marketing*, 39:89–103, 2017.

[YDOT+19] Demet Yesiltepe, Ruth Dalton, Ayse Ozbil Torun, Nick Dalton, Sam Noble, Michael Hornberger, and Hugo Spiers. A wayfinding research in virtual environments: The effect of spatial structure and different conditions on movement. In *12th International Space Syntax Symposium*, 2019.

[YoY19] YoYoGames. Game maker. `https://www.yoyogames.com/gamemaker`, 2019. [Online; accessed July 12, 2019].

[YPY+19] Jianrong Yao, Yanqin Pan, Shuiqing Yang, Yuangao Chen, and Yixiao Li. Detecting fraudulent financial statements for the sustainable development of the socio-economy in china: a multi-analytic approach. *Sustainability*, 11(6):1579, 2019.

[ZDA17] Telmo Zarraonandia, Paloma Diaz, and Ignacio Aedo. Using combinatorial creativity to support end-user design of digital games. *Multimedia Tools and Applications*, 76(6):9073–9098, 2017.

[ZDAR15] Telmo Zarraonandia, Paloma Diaz, Ignacio Aedo, and Mario Rafael Ruiz. Designing educational games through a conceptual model based on rules and scenarios. *Multimedia Tools and Applications*, 74(13):4535–4559, 2015.

[ZHMZ14] Egui Zhu, Arash Hadadgar, Italo Masiello, and Nabil Zary. Augmented reality in healthcare education: an integrative review. *PeerJ*, 2:e469, 2014.