Communication Papers
Media Literacy & Gender Studies

CP, 2020, Vol.9 – No18, pp. 7/23 ISSN 2014-6752. Girona (Catalunya). Universitat de Girona. MARÍN-LORA, C.; CHOVER, M.; REBOLLO, C., y REMOLAR, I.: *A game development environment to make 2D games*. Recibido: 29/12/2019 - Aceptado: 13/05/2020

A game development environment to make 2D games

Autores:
Carlos Marín-Lora cmarin@uji.es
Miguel Chover chover@uji.es
Cristina Rebollo rebollo@uji.es
Inmaculada Remolar remolar@uji.es
Institute of New Imaging Technologies - Universitat Jaume I. Castellón. Spain.

Carlos Marín-Lora
Corresponding Author. E-mail: cmarin@uji.es. Phone Number: +34 *616 29 76 65*

This work has been supported by the Ministry of Science and Technology (TIN2016- 75866-C3-1-R) and the research project of the Jaume I University (UJI-B2018-56). In addition, this work has been possible thanks to the graphic resources created by Kenney from Kenney.nl.

Abstract
The creation of video games involves multidisciplinary processes that are not accessible to the general public. Currently, video game development environments are very powerful tools, but they also require an advanced technical level to even start using them. This article presents a 2D game development environment to propose an alternative model to reduce the technical complexity existing in these systems, presenting a data model and a game editor that allows fulfilling this goal. In order to test its capabilities, several games have been successfully implemented in the proposed environment. With this achievement, it can be stated that it is possible to create video games simply and affordably for the general public without giving up its potential and remarking that there is still a long way to go to reach democratization in the creation of video games and the need to continue working in this field.

1. Introduction

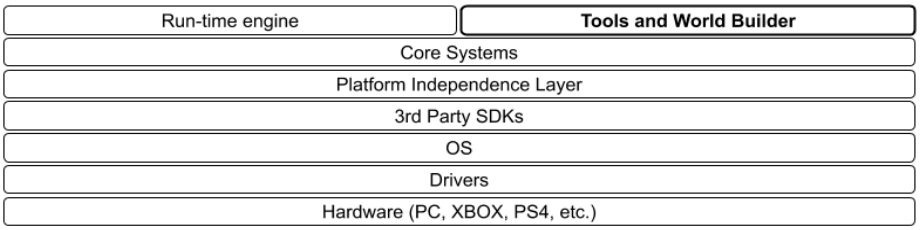
Video game development is a complex process requiring multidisciplinary knowledge and skills, from the artistic and narrative vision to the ability to solve the technological challenges [Blow2004]. Since its first steps as an industry, the developers tried to integrate these processes in order to accelerate production times and optimize resource management [Gregory2014]. From this trend, the game engine concept was born as an environment where compose and implement games regardless of its genre. Currently, game engines are indispensable tools to make commercial games and every professional studio works with one, either proprietary software or third-party.

Certainly, game development has technical requirements that are difficult to meet, but some authors point out other causes such as the lack of a unified language, generic designs or architectures, or how the platform choice influences the development [Anderson2008, Ampaztoglu2010]. These are some of the reasons why it seems necessary to keep researching the processes involved in game development and thus bring them closer to the public without the a priori demanded technical capabilities. In order to meet this need, proposals that alleviate the problematic elements are required.

In this sense, this work draws from the hypothesis that it is possible to reduce the technical complexity of game development environments and without losing any potential to create games. From this assumption, a game development environment has been designed from a reduced data model and a behavior specification system. It starts from a 2D concept to reach the basic elements that define a game as clean as possible.

For the complete fulfillment of this work, the game engine architecture presented by J. Gregory [Gregory2014] has been followed, in which the editing tools are embedded in the environment framework (see Figure 1). However, this paper just addresses the data model and the game editor proposal.

Figure 1.- "Tools built on a framework shared with the game" from J. Gregory [Gregory2014].



In this sense, the features common to the contemporary game engines have been studied and their functionalities and requirements have been analyzed. From this process, a set of requirements has been defined to regulate the design and specification of the proposed game development environment. As a summary, these requirements are presented below:

- A data model based on scenes and game objects known as actors.
- Visual programming based on binary decision trees [Laurent1976, Russell2016].
- Needlessness of loops and complex data structures such as vectors or matrices.

From this point, a game editor has been developed where the composition of scenes and the definition of the actor's behaviors is done through visual elements. Further on this document, the process of implementing two arcade games on the editor will be detailed as demonstrators to verify its validity and show its main characteristics. These games will be described to display the data model and the game mechanics description system defined after the development of this work.

The rest of the document is organized as follows: in section 2, the state of the art about game development environments and its appearance in the literature is presented. Next, in section 3 the environment structure, the data model, and the scene editor and the behavioral rules editor are presented. With this framework, in section 4 two arcade games are developed as demonstrators for this work. Finally, the conclusions of the work will be presented in section 5.

2. State of the art

It is notable that despite the general trend towards the democratization of content creation, it is still complicated to self start with commercial game engines. This problem is especially prominent for the game logic definition since it usually implies certain notions of software engineering [Garlan1993, Ampaztoglu2010] and prior knowledge about general-purpose programming languages and specific game development APIs.

From a theoretical point of view, there is a need to expand the field of research in game development [Lewis2002]. Specifically, Anderson et al. [Anderson2008] highlight the lack of literature in this regard and propose several research lines that should be explored in the future. Some of these proposals are the identification of software components common to all types of games, the establishment of a unified language for game development, the definition of generic video game creation tools, the identification of common elements of all types of games that allow defining an architecture-independent reference and the best practices in game development. In this sense, some works have tried to make contributions to this need trying to redefine the concepts established in the development of games from various perspectives, from programming by components [Folmer2007, Doherty2003], a vision of the Model-View-Controller architecture [Olsson2015] or the introduction of restricted semantics [Tutenel2008] to multi-agent paradigms [Marin-Lora2019, Marin-Lora2020].

Besides that, the analysis of state-of-the-art 2D game development environments shows that some of these alternatives have been already applied. Table 1 shows a summary of the study conducted on game development environments for 2D games with the differences between these systems: their platform, their scripting system, and their behavior specification methodology. The table begins with environments using a visual system like FlowLab [Flowlab2019] and it ends with Unreal [Sanders2016] and Unity [Thorn2019], mostly based on C# and its specific libraries. Some of them, such as Game Maker [GameMaker2019] or RPG Maker [RPGMaker2019], have game logic systems based on scripting tools making them complicated to use. However, others like Stencyl [Stencyl2019] or Construct 2 [Stemkoski2017] rely on visual programming methods such as Scratch [Resnick2009], while Gamesalad [Novak2013], Sploder [Sploder2019] and GDevelop [Correa2015] use their own visual interface.

Table 1.- Behaviour specification classification for the state-of-the-art 2D game engines.

Game Engine	Platform	Scripting method	Behavior specification
Flowlab	Web	Visual scripting	Message passing
Gamesalad	Desktop/Web	Visual scripting	Event-driven
Sploder	Web	Visual scripting	Event-driven
GDevelop	Desktop	Visual scripting	Event-driven
GameMaker	Desktop	GameMaker language	Message passing
Construct 2	Desktop	JavaScript	Event-driven
Stencyl	Desktop	Scratch style	Event-driven
RPG Maker	Desktop	Ruby, C++, Java, JavaScript	Scripting
Unreal	Desktop	C++, BluePrints	Message passing
Unity	Desktop	C#	Scripting

Although these are huge steps forward, its usage still requires a technical profile. Mainly because the encapsulation of scripts in visual elements has been carried out implicitly and without reflecting on its functionality and usability. In fact, there are papers in the current literature indicating how complex to solve a problem can be for a beginner to start through computational techniques [Robins2003, Chang2005, Milne2002] and the assistance that visual programming can provide [Chao2016, Blackwell1996]. Also, different methodologies have been studied to introduce programming concepts, both with traditional coding [Koulouri2015] and with visual programming [Powers2006]. In fact, some authors have carried out experiences associating visual programming and computer games. For example, some authors [Ouahbi2015, Rebollo2018] present a study conducted on programming students to evaluate learning basic programming concepts by creating games, and others [Chen2007] show a study to evaluate a learning methodology for object-oriented programming through video games. On a more specific level, some works have proposed combinations of visual programming methodologies with the elements that a game engine requires to define the behaviors of a game. In this line, Furtado et al. [Furtado2011] propose a description of the game engines based on a more abstract and expressive set of layers. Also, Zarraonandia et al. [Zarraonandia2015, Zarraonandia2017] presented a conceptual model to organize the characteristics of the game in a modular way, where the description and definition required to create a combination of subgames are based on a set of configurable elements and a basic vocabulary for each characteristic. In addition, some software engineering methods have emerged as a possible plan to address this problem, proposing a systematization of the game development process [Reyno2008, Furtado2006]. All this study shows that there is a lot of work in game creation and there is a need to develop new tools to make it accessible for a large number of users.

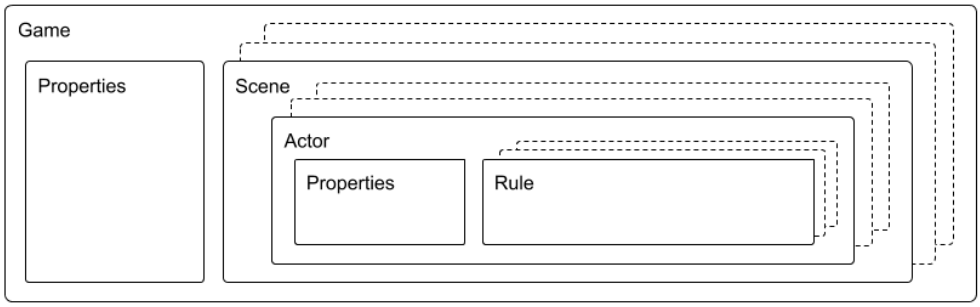
3. Game development environment

In order to frame a proposal to make the game development accessible to the general public, a game development environment is presented based on the definition of its data model, its visual programming elements along with its behavior rules system, and the design of its editor.

3.1. Data model

In this work, a game is composed of a set of properties and a list of scenes. A scene is made of a list of actors that are composed of a set of properties and a list of behavior rules. A diagram representing this organizational structure is drawn in Figure 2.

Figure 2.- Structure diagram for the game development environment



The game's properties include parameters that control the scene rendering or physical behaviors, among others. More generally, these properties can be classified as follows:

- **Camera:** Includes information about the geometric transformations of the camera about the game.
- **Audio:** Variables with which to define and modulate the sounds of the game based on parameters such as bread, volume, and loop.
- **Physics:** Parameters to establish the intensity of gravity in the game, in case a game with realistic physics is required.
- **New:** Variables added by the game designer to meet a specific need.

Besides that, each game scene represents an independent stage of the game, to be employed as convenient in the game design.

The actors are the main and unique elements for the scene's composition. They are responsible for representing any game element and executing the game logic, and, as well as the game, have a set of properties that define them. In addition, they can acquire new properties, to meet the game needs. These properties can be grouped as follows:

- **Geometry:** Information related to the geometric transformations of the actors.
- **Render:** properties on the visual appearance of the actor including its image, opacity, and color tinting, among others.
- **Text:** In addition, actors can represent text on the screen according to the font, size, color, and style properties. In addition, they can show property values, both their own and those of other actors or the game.
- **Audio:** In the same way as in the game, actors can play sounds based on the same properties.
- **Physical:** If the actor is physical, their speed and material properties are activated: density, friction, and damping.
- **New:** In addition, actors can incorporate new properties to expand their capabilities.

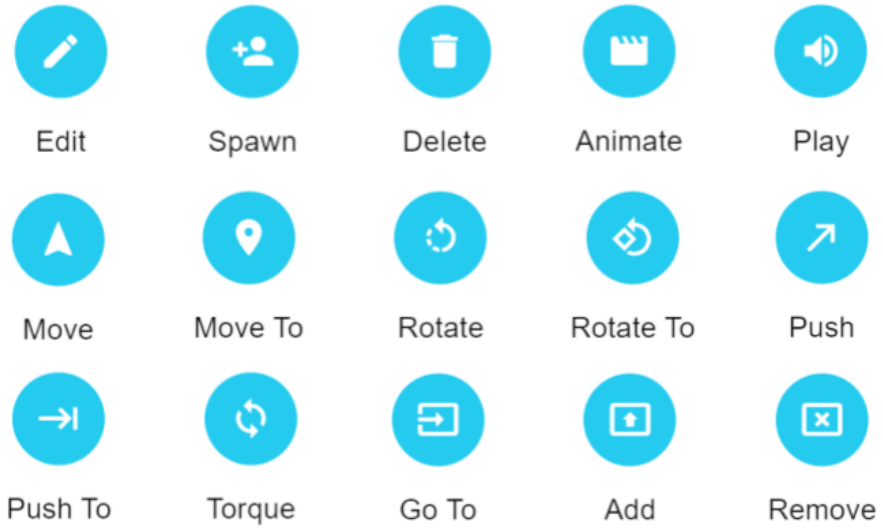
3.2. Visual programming

Actors are also responsible for defining the game logic from a model of rules of behavior. This model is based on first-order logic [Ligeza2006, Brachman1992] and inspired by a multi-agent paradigm [Wooldridge2009, Marin-Lora2020]. A behavior rule is determined by binary decision trees [Millington2009] driven by a reduced set of actions and conditions, ready to execute if the flow passes through them as conditions are met or not. Both actions and conditions work with arithmetic expressions and mathematical functions: *sin*, *cos*, *tan*, *asin*, *acos*, *atan*, *sqrt*, *random*, etc; and with numbers and booleans data types. These elements can provide basic coding knowledge without sacrificing the complex development of the game, considering only that the game loop implements the evaluation of the behavior of each actor in each iteration. Furthermore, boolean expressions and complex data structures such as matrices, matrices, or other complex structures such as trees or graphics to create actor rules have been ruled out, since they are not necessary for this architecture.

The actions are the elements of the game logic that give rise to the behaviors of the actors, in fact, its operation is based on the modification of properties, either on the game or other actors. After a review of the behaviors that commonly implement these actions in games, a reduced set of actions has been organized. From this set, the game designer must compose his logic. The study has resulted in the fifteen actions presented in Figure 3, some of which are described below:

- **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.
- **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.
- **Move:** Specialisation of the update operation that applies a displacement on the game object by an angle and 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.

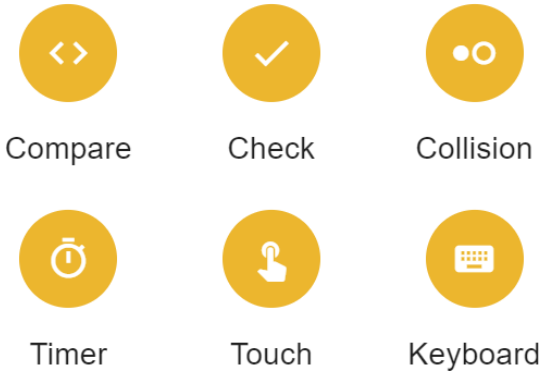
Figure 3.- Visual programming actions.



Actions usually obey events that trigger them. In the proposed system, these events are called conditions and are the elements with which the logic system conducts the execution flow of the actions. A condition basically consists of a boolean expression that determines a logical relationship between system properties and/or arithmetic expressions. To accomplish this task, the same analysis process performed in the previous case has been carried out, resulting in a reduced set of conditions displayed in Figure 4 that include the following types:

- **Check**: Evaluation of a boolean property from the game or a game object.
- **Compare**: Relational condition between a property with an arithmetic expression in modes *greater*, *greater-equal*, *equal*, *less-equal,* or *less*.
- **Pointer**: Pointer events managed by the input system in terms of *down*, *move,* and *up* events.
- **Keyboard**: Keystrokes condition for a key in modes such as *down*, *up,* and *pressed*.
- **Collision**: Collision detection between game objects based on tags.
- **Timer**: Temporary condition where a system timer is compared with a cut-off time. It is false if the timer is below the cut-off time. When true, the timer is restarted.

Figure 4.- Visual programming conditions.

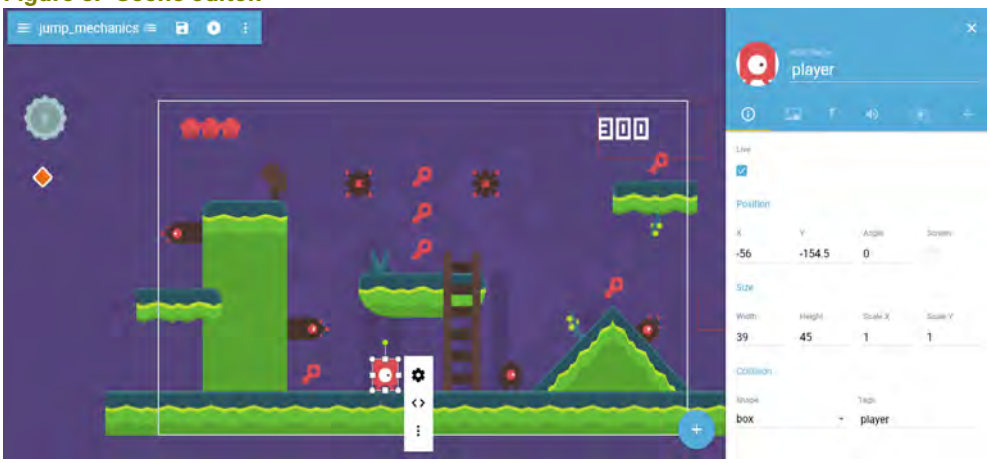


3.3. Editor

In order to facilitate the use of the data model presented in the previous section, and to enable the creation of games with this system, a game editor proposal is presented below. One of the goals of this work is to reduce the level of complexity inherent in game development environments, and for this, it is necessary to identify the simplest and most accessible methods with which to group game development tools. The development environment consists of two integrated editors: a scene editor and a rule editor. Both editors are based on a paradigm of visual composition with functionalities such as drag and drop, geometric transformations using gizmos, and access to properties for consultation or modification. The scene editor is accessed directly at the application start, but the rule editor is activated from the interface, by selecting the actors' rules button. At a specification level, the complete environment has been developed by adopting concepts of user interface and interaction of slide-show applications [Tufte2003]. This approach is due to the fact that these applications are usually easy to use and oriented to non-technical people, and also have similarities with game elements: slides such as scenes, object positioning and property editing. Besides that, the design of the editor has been based on the Google Material Design specification [GoogleDesign2019], where its definition is oriented to multi-device applications with fluid navigation.

An example of the scene editor interface is presented in Figure 5, where the canvas is filled with actors, a blue button to create a new actor, and two interface menus. On the one hand, in the upper left corner, the options for the general control of the game such as the scene list or the game-saving are arranged, on the other hand, in the right side, a panel displays the properties of the selected actor. Also, when selecting an actor, the actor's transformation gizmos are displayed along with a modular white menu located next to the red character shown in Figure 5. From this menu, the user can access the actor's properties, the actor's rules, and some functions such as copy and paste or removal.

Figure 5.- Scene editor.



The actor's logic is defined in the rule editor, which is based on binary decision trees and where to arrange actions and conditions until the desired behavior is fulfilled. To edit the rule, the user must fill in the decision tree through actions and conditions. These logical elements are available from the *If* and *Do* buttons, which act as shortcuts to the sets of conditions and actions available in the system, respectively. Figure 6 displays a *jump* mechanic rule in the rule editor, also, and in order to facilitate the understanding of the rule, a version of pseudocode is attached in Algorithm 1. This rule controls the jump of the character, and it waits for a keyboard condition and a collision with the ground condition. When these two conditions are met, the flow will travel through the right branch and an action will be applied that edits the velocity property on the Y-axis and sets it to 350 units. If either of these two conditions is not met, the flow will travel through the branches on the left side and no action will happen

Algorithm 1.- *Example of the associated pseudocode for the rule in Figure 3.*

```
If ( collision( ground ) )
        If ( keyboard( spaceKey, down ) )
                edit( velocity_y, 350 )
        End
End
```

Figure 6.- Rule editor

4. Results

Next, in order to display the features of this work, the document introduces two arcade games implemented in the game development environment described in the previous section as demonstrators. The description of these games intends to display the data model and the game mechanics description system derived from the development of this work. The methodology followed to define the system was based on developing as many different game mechanics and as many different genres as possible, trying to generate an extended knowledge about the basic actions required to compose the games. As the study went forward, some required elements were added while redundant or unnecessary elements were removed. Finally, a robust method was reached through which all the games proposed so far have been completed, supporting that it is possible to reduce the technical complexity level necessary to create games without losing potential.

4.1. Asteroids

Asteroids is an arcade shooting game published by Atari in 1976 in which the player drives a spaceship that goes through an asteroid belt. The goal of the game is to shoot and destroy as many asteroids as possible and not collide with them. As the game progresses, the number of asteroids increases, increasing the degree of difficulty. Figure 7 shows a screenshot of the game implemented in this environment during its execution.

For the implementation of this game, the degree of difficulty increases as the game progresses, and it is controlled by a new property initialized to 1. The actors for this implementation are spaceship, asteroids, and projectiles. The first always present on the screen, and the last two instantiated when necessary. In addition, in a more specific way, two specializations of asteroid actors have been created with the aim of representing their subdivision levels: *AsteroidBig* and *AsteroidSmall*, where both actors are initialized with random direction, linear velocity, and angular velocity. The asteroids and the spaceship change to their opposite position on the screen with inverse speed when they reach one of the limits of the scene. On the other hand, projectiles are created by the ship with a constant velocity vector directed from the orientation of the ship at the time of its spawn. The implementation of two of its mechanics is detailed as a behavior rule:

Figure 7.- Asteroids game.

Player movement: To transfer user actions to the player's actor, the spaceship, a rule must be set in the actor. In this game, the player controls the movement of the ship by pushing and rotating the ship. To control this, three events must be controlled: positive rotation, negative rotation, and thrust. The *thrust* will be carried out based on the rotation of the ship and an impulse parameter determined by a new property. The resulting rule is defined as pseudocode in Algorithm 2.

Algorithm 2.- *Player movement rule.*

```
If ( keyboard( leftKey, down ) )
        edit( rotation, rotation - 1 )
End
If ( keyboard( rightKey, down ) )
        edit( rotation, rotation + 1 )
End
If ( keyboard( upKey, down ) )
        push( thrust, rotation )
End
```

Asteroid Subdivision: The asteroid subdivision is implemented in one of the *AsteriodBig* actors' rules set. It starts from collision events with a *Projectile* actor and causes its removal from the scene along with the spawning of three *AsteroidSmall* actors. The pseudocode of this rule is shown in Algorithm 3.

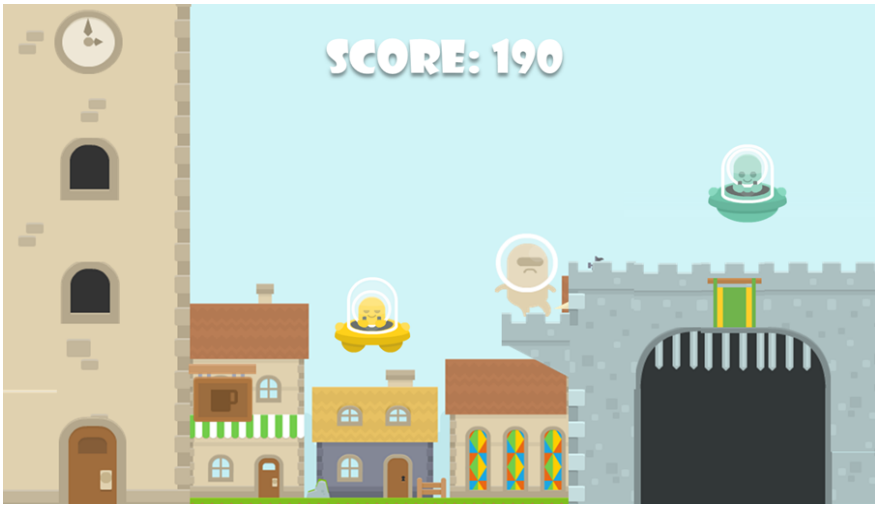
Algorithm 3.- *Asteroid subdivision rule.*

```
If ( collision( Projectile ) )
        spawn( AsteroidSmall )
        spawn( AsteroidSmall )
        spawn( AsteroidSmall )
        destroy( )
End
```

4.2. Tower Bridge Defense

The other game to be developed in this work is inspired by the London's Tower Bridge Defense [TBD2019] platform game, used by Unity as a tutorial for creating 2D games on its platform. The game is framed in the context of an invasion of aliens that the player must destroy within a scenario with five platforms. The game starts with the player-controlled character on one of the upper platforms, and with aliens falling from the top of the screen. The player can move around the stage, avoiding colliding with the aliens and shooting them to eliminate them and accumulate as many points as possible while remaining alive. A screenshot of the game, while it is running on the development environment, is displayed in Figure 8.

Figure 8.- Tower Bridge Defense game.



The description of this game has been focused on the shooting mechanics and the alien's autonomous movement since the movement of the player is quite similar to the Asteroids case and it seems more useful to describe the system capabilities with different mechanics.

- **Player Shooting**: The shots that eliminate the aliens are generated from the spawning of *Bullet* actors activated by keyboard events, this mechanic is described in Algorithm 4. These bullets move at a constant speed and in the orientation of the player until they are destroyed either by collision with an alien or with the limits of the screen

Algorithm 4.- *Player shooting rule.*

```
If ( keyboard ( space, down ) )
        spawn ( Bullet )
End
```

- **Alien Movement**: So far, the user's orders have been connected with the actors through rules. At this point, it is necessary to define the movement of the *Alien* actors without external interaction, in other words, autonomously. This movement has been defined by a *thrust* property and two behavior rules, the first dependent on the moment in which they come into contact with the platforms when they randomly choose a direction, and the second, which causes a direction switch when the stage limits are reached. The pseudocode of these rules can be seen in Algorithms 5 and 6, respectively.

Algorithm 5.- *Alien initial movement rule.*

```
If ( collision( Platform ) )
        If ( check( alien.first ) )
                edit( alien.velocityX, alien.thrust * ( 1 – 2 * random( 0, 1 ) ) )
                edit( alien.first, false )
        End
End
```

Algorithm 6.- *Alien's boundaries switch.*

```
If ( collision( Boundaries ) )
        edit( alien.velocityX, alien.velocityX * –1 )
End
```

5. Conclusions

In this work, an environment for the creation and development of video games has been presented. Several attempts have been made to identify the mechanisms that define the creation of games, and their integration into an environment of composition and visual programming has proceeded. All without relying on hierarchical scene structures, complex data structures such as matrices or vectors, and repetition loops, assigning the game definition to the game elements known as actors. This concept gives greater specific weight to the actors, through which any element of the game and its mechanics are determined. To define the mechanics of the actors, a visual rule editor based on binary decision trees has been designed, through which the actors' logic and, therefore, the game logic is conducted.
After defining the data model and the editor that drives it, and in order to demonstrate its potential, several 2D arcade games have been implemented. In this document two of them have been presented, through which the tools provided by the editor and the possibilities offered are perceived.

The proposed environment has proved efficient in its task of creating games based on the designed data model and the implemented editor. In addition, it is obtained that it is necessary to continue working in the field of video game development, emphasizing the processes of definition and specification of game mechanics, especially in order to introduce non-technical personnel to the sector.

The next steps in this project are to explore the potential of the data model, the game logic system, and the usability of the editor. Including the definition of a language for the specification of games and the extension of the environment to a 3D version.

Acknowledgments

This work has been supported by the Ministry of Science and Technology (TIN2016- 75866-C3-1-R) and the research project of the Jaume I University (UJI-B2018-56). In addition, this work has been possible thanks to the graphic resources created by Kenney from Kenney.nl.

References

[Ampatzoglou2010] Ampatzoglou, A., & Stamelos, I. (2010). Software engineering research for computer games: A systematic review. Information and Software Technology, 52(9), 888-901.

[Anderson2008] Anderson, E. F., Engel, S., McLoughlin, L., & Comninos, P. (2008). The case for research in game engine architecture.

[Blackwell1996] Blackwell, A. F. (1996, September). Metacognitive theories of visual programming: what do we think we are doing?. In Proceedings 1996 IEEE Symposium on Visual Languages (pp. 240-246). IEEE.

[Blow2004] Blow, J. (2004). Game development: Harder than you think. Queue, 1(10), 28.

[Brachman1992] Brachman, R. J., Levesque, H. J., & Reiter, R. (Eds.). (1992). Knowledge representation. MIT press.

[Chang2005] Chang, S. E. (2005). Computer anxiety and perception of task complexity in learning programming-related skills. Computers in Human Behavior, 21(5), 713-728.

[Chao2016] Chao, P. Y. (2016). Exploring students' computational practice, design and performance of problem-solving through a visual programming environment. Computers & Education, 95, 202-215.

[Chen2007] Chen, W. K., & Cheng, Y. C. (2007). Teaching object-oriented programming laboratory with computer game programming. IEEE Transactions on Education, 50(3), 197-203.

[Correa2015] Correa, J. D. C. (2015). Digitopolis II: Creación de videojuegos con GDevelop. Jose David Cuartas Correa.

[Doherty2003] Doherty, M. (2003). A software architecture for games. University of the Pacific Department of Computer Science Research and Project Journal (RAPJ), 1(1).

[Flowlab2019] Flowlab. https://flowlab.io/ [Online; Last accessed: 2019-11-20] (November 2019).

[Folmer2007] Folmer, E. (2007, July). Component-Based Game Development–A Solution to Escalating Costs and Expanding Deadlines?. In International Symposium on Component-Based Software Engineering (pp. 66-73). Springer, Berlin, Heidelberg.

[Furtado2006] Furtado, A. W., & Santos, A. L. (2006, October). Using domain-specific modelling towards computer games development industrialization. In The 6th OOPSLA workshop on domain-specific modelling (DSM06).

[Furtado2011] Furtado, A. W., Santos, A. L., Ramalho, G. L., & de Almeida, E. S. (2011). Improving digital game development with software product lines. IEEE Software, 28(5), 30-37.

[GameMaker2019] Game Maker. YoYo Games. http://www.yoyogames.com [Online; Last accessed: 2019-11-20] (November 2019).

[Garlan1993] Garlan, D., & Shaw, M. (1993). An introduction to software architecture. In Advances in software engineering and knowledge engineering (pp. 1-39).

[GoogleMaterial2019] Google Material Design. https://design.google [Online; Last accessed: 2019-11-20] (November 2019).

[Gregory2014] Gregory, J. (2014). Game engine architecture.

[Koulouri2015] Koulouri, T., Lauria, S., & Macredie, R. D. (2015). Teaching introductory programming: A quantitative evaluation of different approaches. ACM Transactions on Computing Education (TOCE), 14(4), 26.

[Laurent1976] Laurent, H., & Rivest, R. L. (1976). Constructing optimal binary decision trees is NP-complete. Information processing letters, 5(1), 15-17.

[Lewis2002] Lewis, M., & Jacobson, J. (2002). Game engines. Communications of the ACM, 45(1), 27.

[Ligeza2006] Ligeza, A. (2006). Logical foundations for rule-based systems (Vol. 11). Heidelberg: Springer.

[Marin-Lora2019] Marin-Lora, C., Chover, M., Sotoca, J. M. (2019). Prototyping a Game Engine Architecture as a Multi-Agent System. 27th International Conference in Central Europe on Computer Graphics, Visualization and Computer Vision 2019 (WSCG 2019).

[Marin-Lora2020] Marin-Lora, C., Chover, M., Sotoca, J. M., & García, L. A. (2020). A game engine to make games as multi-agent systems. Advances in Engineering Software, 140, 102732.

[Millington2009] Millington, I. (2009). AI for Games. CRC Press.

[Milne2002] Milne, I., & Rowe, G. (2002). Difficulties in learning and teaching programming—views of students and tutors. Education and Information Technologies, 7(1), 55-66.

[Novak2013] Novak, J. (2013). The Official GameSalad Guide to Game Development. Cengage Learning.

[Olsson2015] Ollsson, T., Toll, D., Wingkvist, A., & Ericsson, M. (2015, May). Evolution and evaluation of the model-view-controller architecture in games. In 2015 IEEE/ACM 4th International Workshop on Games and Software Engineering (pp. 8-14). IEEE.

[Ouahbi2015] Ouahbi, I., Kaddari, F., Darhmaoui, H., Elachqar, A., & Lahmine, S. (2015). Learning basic programming concepts by creating games with Scratch programming environment. Procedia-Social and Behavioral Sciences, 191, 1479-1482.

[Powers2006] Powers, K., Gross, P., Cooper, S., McNally, M., Goldman, K. J., Proulx, V., & Carlisle, M. (2006, March). Tools for teaching introductory programming: what works?. In ACM SIGCSE Bulletin (Vol. 38, No. 1, pp. 560-561). ACM.

[Rebollo2018] Rebollo, C., Marín-Lora, C., Remolar, I. & Chover, M. (2018). Gamesonomy Vs Scratch: Two Different Ways To Introduce Programming. 15th International Conference On Cognition And Exploratory Learning In The Digital Age (CELDA 2018). Ed. IADIS Press. ISBN 9789898533814.

[Resnick2009] Resnick, M., Maloney, J., Monroy-Hernández, A., Rusk, N., Eastmond, E., Brennan, K., ... & Kafai, Y. B. (2009). Scratch: Programming for all. Commun. Acm, 52(11), 60-67.

[Reyno2008] Reyno, E. M., & Cubel, J. Á. C. (2008). Model-Driven Game Development: 2D Platform Game Prototyping. In GAMEON (pp. 5-7).

[Robins2003] Robins, A., Rountree, J., & Rountree, N. (2003). Learning and teaching programming: A review and discussion. Computer science education, 13(2), 137-172.

[RPGMaker2019] RPG Maker. https://www.rpgmakerweb.com [Online; Last accessed: 2019-11-20] (November 2019).

[Russell2016] Russell, S. J., & Norvig, P. (2016). Artificial intelligence: a modern approach. Malaysia; Pearson Education Limited.

[Sanders2016] Sanders, A. (2016). An Introduction to Unreal Engine 4. AK Peters/CRC Press.

[Sploder2019] Sploder. http://www.sploder.com [Online; Last accessed: 2019-11-20] (November 2019).

[Stemkoski2017] Stemkoski, L., & Leider, E. (2017). Game Development with Construct 2: From Design to Realization. Apress.

[Stencyl2019] Stencyl. http://www.stencyl.com [Online; Last accessed: 2019-11-20] (November 2019).

[TBD2019] Unity Technologies. Unity Tower Bridge Defense Tutorial. (2019). https://learn.unity.com/tutorial/unity-for-2d-new-workflows-in-unity-4-3. [Online; Last accessed: 2019-11-20] (November 2019).

[Thorn2019] Thorn, A., Doran, J. P., Zucconi, A., & Palacios, J. (2019). Complete Unity 2018 Game Development: Explore techniques to build 2D/3D applications using real-world examples. Packt Publishing Ltd.

[Tutenel2008] Tutenel, T., Bidarra, R., Smelik, R. M., & Kraker, K. J. D. (2008). The role of semantics in games and simulations. Computers in Entertainment (CIE), 6(4), 57.

[Tufte2003] Tufte, E. R. (2003). The cognitive style of PowerPoint (Vol. 2006). Cheshire, CT: Graphics Press.

[Wooldridge2009] Wooldridge, M. (2009). An introduction to multiagent systems. John Wiley & Sons.

[Zarraonandia2015] Zarraonandia, T., Diaz, P., Aedo, I., & Ruiz, M. R. (2015). Designing educational games through a conceptual model based on rules and scenarios. Multimedia Tools and Applications, 74(13), 4535-4559.

[Zarraonandia2017] Zarraonandia, T., Diaz, P., & Aedo, I. (2017). Using combinatorial creativity to support end-user design of digital games. Multimedia Tools and Applications, 76(6), 9073-9098.

CURRICULUM VITAE

Carlos Marin-Lora is a Ph.D. student in Computer Science at Jaume I the University of Castellón. He holds a degree in Multimedia Engineering from the University of Valencia in 2015 and a master in Intelligent Systems from the Jaume I the University. His research interest includes computer graphics and multimedia, game logic and game engines, artificial intelligence, behavior specification, pattern recognition, and web systems. He also works as Character and Creature FX artist for animation companies such as Ilion Animation Studios and Lightbox Studio.

Miguel Chover is Full Professor in the Department of Computer Languages and Systems at Jaume I University of Castellón. He received his PhD in computer science in the Polytechnic University of Valencia in 1996. He is currently director of the Center for Interactive Visualization and member of the Institute of New Imaging Technologies of Jaume University I. His research lines include: geometric modeling, interactive visualization and video game technology. He is an active member of the Spanish Computer Graphics Association EUROGRAPHICS S.E and member of the executive committee of the Spanish Society for Video Game Sciences (SECiVi).

Cristina Rebollo is a professor at the Universitat Jaume I of Castellón in Spain. She received her MS degree in Computer Science in 1988 from the University of Deusto of Bilbao, Spain. She received her Ph.D. in Computer Science from the Universitat Jaume I of Castellón, Spain in 2006. She is currently working at the Department of Computer Languages and Systems at the University Jaume I. Her research areas include multiresolution modeling, real-time visualization, video games, and virtual and augmented reality.

Inmaculada Remolar received her Ph.D. in Computer Science at the Universitat Jaume I in 2005. Currently, she is director of the Institute of New Imaging Technologies and associate professor at the Computer Languages and Systems Department at the Universitat Jaume I since 2008. Her research interests include geometric modeling, interactive visualization, and videogame technologies. In reference to these subjects, she has participated in numerous research projects, being IP of some of them. She has also participated in numerous projects with companies, where she has been involved in the application of the research to the business fields.