Game Development and Evaluation of the EvoGlimpse Video Game

Bianca-Cerasela-Zelia Blaga[1], Dorian Gorgan[2]
Technical University of Cluj-Napoca
Computer Science Department
[1]*E-mail: zelia.blaga@cs.utcluj.ro*
[2]*E-mail: dorian.gorgan@cs.utcluj.ro*

Abstract. Game development is a complex task that requires a lot of hard work and patience because it contains various elements such as 3D objects, collision detection, scripting, sound management, animation, rendering, control, and artificial intelligence. Video games are also interactive applications; therefore, they need to be designed in such a way that would enable a high level of usability. In this paper, the development methodology steps that were done for creating a video game are presented. We also propose a heuristic evaluation with the purpose of answering questions that can determine if the game respects the usability requirements. The goal is to gain knowledge in game development with a hands-on experiment and to estimate the level of usability of the final product.

Keywords: development methodology, heuristic evaluation, game design, game implementation, video games, usability.

1. Introduction

It has been more than 40 years since digital games have been around. They require interaction between a human and a digital device. Video games were defined by Salen (2003) as "a system in which players engage in artificial conflict, defined by rules, that results in a quantifiable outcome". The word "video" refers to Cathode Ray Tube (CRT) monitors, and it is still used nowadays even though games can be played on a variety of devices that have nothing to do with CRT. Nowadays, video games have become a medium to present narratives and sometimes they are even compared to movies. They present a vision that the creator has on a world in which players can immerse into.

The game development methodology determines the quality of the final product. Emphasis is put on how the game looks and feels (i.e. the user interface), therefore this affects the source code, as we will see in this paper too. Game development is a visual technique, because the implementation is

done first, and then the system is tested to see if the demands are met. The next step is decided based on the requirements and how it looks, with the iteration cycles being very short.

The main motivation behind the concept of usability of an interactive application, so implicitly of video games, relies on its capacity of establishing the success or failure rate of a software product. The evaluation of usability can be done during the implementation and design stages, which is highly recommended. Evaluation is an iterative process, by intercalating it in the stages of development, and because it has the advantage of highlighting the design and implementation flaws, errors, and specific deficiencies, which can only be observed during testing.

This paper is structured as follows: in Section 2 will be presented a literature review of this domain, together with some video game relevant concepts. The chosen steps are briefed in Section 3. In Section 4 topic selection is discussed – this includes what inspired the game creation, what the plot is, and what are the final specifications that will be implemented; then a low-fidelity game prototyping method is employed to establish how the interface and the controls of the game should look like and function; finally, details of game design are presented – scenarios and actions, while in Section 5, the actual implementation steps that were done to create the system are discussed, together with some results which are presented in Section 6. The final observations are made in Section 7. An in-game screen-shot can be seen in Figure 1.

Figure 1. The game interface of EvoGlimpse

2. Related Work

Game development is a set of complex processes that need knowledge from multiple domains, such as software engineering, visual arts, graphics design, modeling, simulations, psychology to name just a few disciplines. Therefore, it becomes obvious that planning and managing the development of such a project requires using clear development methodologies and continuous evaluation. In the paper written by Serdar (2016), the aspects of game development are presented in detail, and emphasis is put on the necessity of having an explicit, well-defined structure when developing a digital game.

First, an immersive game story has to be established. Then the main world is built upon it, with the principal purpose of entertaining the target audience (Adams, 2009). The world should be kept simple and elegant, it has to fit the story, and should be logically consistent. The last one means that the gameplay should not break the game logic (e.g. physics), and should allow smooth transitions, without distractions.

Then, the developers decide the main gameplay mode, which is the stage in which a player will spend most of his/her time into. Then the world scene is decided, which can be 2D, 2.5D or 3D. Afterward, interactions between the player and the game scene and objects are defined. These should be functionally correct and suitable for the genre. For example, the developer has to take into account what kind of input devices (mouse, keyboard, joystick etc.) will be used by the player and adjust the interactions to them.

In the next stage, the gameplay is established. This contains what levels the game has, the challenges for each one of them, how players can gain experience and level up, and how they can lose or gain points. The ways of winning and losing the game should be clearly established too. Each game needs a well-designed, believable and realistic character that would fit the game story and that would motivate the players. The success of the game depends on how deep and lovely the character is, the level of engagement it gets from the players, and how easy it is to connect with it. As it is stated in the paper written by Bernhardt (2011), the goal is to create situations and characters that players can truly engage and love. Also, the game should allow character customization, to make the gameplay more personal.

Afterward, the core game mechanics are defined. These are the interactions that occur frequently. For example, firing a weapon would be the main mechanic of a shooter game. This further allows to build around it challenges and actions in the gameplay. Also, the game genre is defined. This is used to classify games based on the interaction and gameplay

techniques. A list of commonly used video game genres are presented in the articles of Sellers (2005) and Pinelle et al. (2008).

In the iterative stages of game development, the design goes from general to more specific, as the developers elaborate the plot ideas. Designers define the story, game worlds, characters, gameplay modes, core game mechanics, levels etc. They get into more details by creating a prototype to check how the ideas fit together and how they actually look and fell in a real environment. Prototyping is used to visually represent the interface of the game, so potential players can experiment with the expected functionalities under virtually simulated conditions. It is fast and cheap to build, and also easily modifiable. The main purpose of prototyping is to discover flaws of game design and to give feedback to the developers.

The last stage of the iterative development methodology is the evaluation. This can be both functional – test the correctness of the functionality of the controls and interactions, and heuristic – establish a set of criteria and rules for testing. The game quality is thus assessed, and the errors are identified. Based on the results' analysis, solutions are proposed to improve the game.

To connect all the previously mentioned elements of the game development, such as characters, levels, game world, gameplay, mechanics etc., a flowboard technique is used. This has the advantage of being non-linear, and it contains scenes linked to each other by arrows. All this development methodology and especially the conclusions and details that the developers came up with, should be recorded in the Game Design Specification Document. This is constantly updated and maintained, being the blueprint that describes every aspect of the game (Mitchell, 2012).

Even though the game industry had been rising, being a successful developer is getting more difficult (Bethke, 2003). The digital game has to constantly be revised and updated to keep up with the desires of customers. That is why having project management is crucial because then the manager is responsible for the stages of development like planning, execution, resource and people management and he/she makes sure that the objectives are met.

3. Game Development Methodology

In this paper, the actual steps that were put in place to create a game are:

1. **Analysis:**
   - ***1.1.Topic selection*** – in which the inspiration sources are presented;

1.2. ***Game specifications*** – this includes user requirements and usability specifications;
1.3. ***Prototyping*** – which helps explore different solutions by creating low-fidelity user interfaces;
1.4. ***Scenario and task description*** – where the final components of the game are established; steps to establish how the game should be played, how it can be lost or won, interactions of the player with objects from the scene etc.

2. **Implementation:**
   2.1. ***Tools*** – the software that was used in building the actual game;
   2.2. ***Game objects*** – the 3D models that compose the game world;
   2.3. ***Implementation details*** – how the scenarios and the actions were brought to life;
3. **Evaluation –** results of the functional and heuristic evaluation, together with solutions to the problems that were found, and future development possibilities.

4. Analysis

4.1. Topic selection

To keep the players interested, the game should have a story. This also motivates the users to want to go out and reach the next level. For example, there can be transitional scenes between levels. Also, the story can be either linear or non-linear. A great example of the latter is "Life is strange" (Enix, 2015), where time travel is one of its main themes to show how your choices affect the lives of others. In this section, the main sources of inspiration for the developed game will be shown, together with the plot of a complex game, and what will actually be implemented.

4.2. Inspiration

The main source of inspiration for the game comes from "2001: A space odyssey" (Clarke, 1968) , that stands out for the evocative power it has, even though it relies only on a small set of resources and songs. Humans have always been fascinated with how the world has evolved and have tried to come up with understandings of how the knowledge was found out. Here, the source of knowledge is the monolith – it can be seen in Figure 2, which apparently is only a big black block of unknown matter, but which was

actually placed in our world by unknown beings to provide guidance and survival ideas. The movie and the book were created in 1968, before the man first walked on the moon, and has such powerful visionary scenes that are relevant even to this day.

Other great visual inspiration sources are “Blade Runner 2049” (Villeneuve, 2017) – for the vision about the future of our world, and also the smooth movement of the flying cars, and “Dunkirk” (Nolan, 2017) – for the scenes where the planes fly over the water. Screenshots from the two movies can be seen in Figure 3 and Figure 4. All the previously mentioned movies have powerful cinematography and soundtracks. Another main inspiration theme, the book “The Greatest Show on Earth” by Richard Dawkins (Dawkins, 2009) provides exhaustive information about evolution, starting from the early stages of life until our days and the current discoveries that man has made in this area.

Figure 2. The first discovery on the monolith by the apes

Figure 3. Screenshot from Blade Runner 2049

Figure 4. Screenshot from Dunkirk

4.3. Game plot

EvoGlimpse aims to give players a glimpse into evolution from the perspective of an exterior observer, who can travel at different points in time of Earth's existence. This game is heavily inspired by the movie and the book "2001: A space odyssey" (Clarke, 1968) , in which a civilization of advanced beings helps humans that are in different stages of evolution by presenting to them ways that can aid in their survival.

A series of worlds would be available, starting from the first appearance of life – the fusion between RNA and an enzyme, then at different stages of the evolution of species – underwater life, transitioning to earth land and dinosaurs, continuing with the human history – from the ancestors until today, and for a plus of entertainment, will continue with a science fiction view of humankind – the union of human-machine and the exploration of the universe. The player would be able to travel in these worlds in different specific shapes: atoms, energy, swimming, walking, riding animals, driving the car, flying the flying cars, and exploring the outer space in spaceships.

Each phase has as objective finding the knowledge source, represented by the monolith, which has an imposing shape, tall, black, created by a superior entity and which holds superior information about the current state of the world. For example, in stone age, this can offer to the monkeys the idea of creating weapons that represent an advantage in the fight for survival. As a world is explored, different obstacles appear, and the player must overcome them with the current set of skills. This is enhanced each time the monolith is found. Once the world has been completely observed and the enemies are defeated, the monolith appears to present the way of going from the past to the future. Using visual and auditory information, the player will know if he/she is close to the location of the monolith, and when this will be found, an educational video about evolution will be presented. The player will be able to see all finished phases and all the discovered videos in a library, to which he/she can return at any time.

4.4. Game Specifications

For the actual game implementation, the goal was to create only a world, a futuristic one, on a planet covered by water, in a developed society, with modern architecture and flying cars. The main enemies will be planes guided by artificial intelligence. The player will have to protect itself from them by shooting, for example with bullets, plasma, or laser. The main plot of the game follows 3 stages. In the first one, the player will have some time to get used to the planet and the controls, being able to peacefully explore

and observe the world scene. In the second stage, the player will have to protect the planet from some invaders; as the game advances, the abilities increase. In the last stage, since an advanced technology state has been reached, the monolith will appear in an unknown location and will have to be found by following its sound signals. Therefore, this is a shooter action game, in which the main action is firing weapons. This genre is most suitable for ages 12 and over because it does not contain unnecessary violence and inappropriate scenes for young players.

4.5. Prototyping

A low-fidelity game interface has been created, which can be seen in Figure 5. The main interface of the game will have useful information for the player but will be mainly dedicated to presenting the game scene of the world and the vehicle. The game scene prototype can be seen in Figure 6; the focus is on creating an aesthetic 3D world and a realistic vehicle for the player to control. The components of the game interface are:

1. The main game scene. Here the movement of the car can be observed. The perspective of the camera can be changed to allow the player to see further away.
2. The menu contains 5 buttons, each one of them taking the player in a different configuration option. The menu interface can be seen in Figure 7.
3. Relevant logs for the game. After some options are selected in the menu, a confirmation message will be displayed.
4. Information about the current state of the game, like health, attack speed, and armor. In a window will be displayed the number of enemies remaining and the number of enemies that were taken down.
5. The map of the world and the position of the player in it, in the form of a birds-eye view.

The player could access the menu to change the game settings. By pressing the Menu button, the player has access to 4 options. Pressing the corresponding buttons, the interface will change for the desired modifications to be made. Swapping the car can be done in the menu by pressing the button Select car. This interface can be seen in Figure 8. On the screen, the available options will be shown. By moving the mouse over one

of them, this will receive a yellow glow. If the player clicks in the vehicle area, it will receive a green glow.

To select a type of attack, press the button Select attack type from the menu. The options will be shown; these will vary from bullets to laser to plasma. Pressing on one of them will select it, and the user will be returned to the main game scene. The volume could be changed by pressing the button Sound settings from the menu, where a slider will enable the player to modify the sound volume. The changes will be heard in real time because music will play. The game can be saved by pressing the Save button. The game can be paused by pressing the key P. The player can exist from the game by pressing the Escape button.

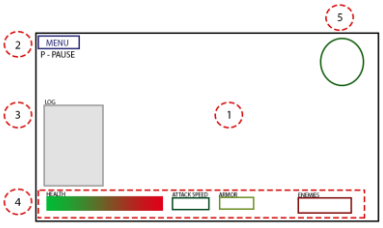


Figure 5. Game interface prototype


Figure 6. Game scene prototype

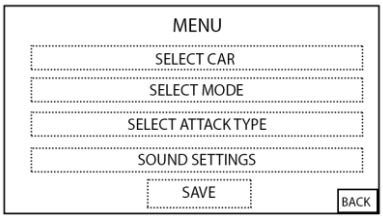


Figure 7. Menu prototype

Figure 8. Vehicle selection menu prototype

4.6. Scenario and task description

In this stage of the game development, we establish what we want to actually have in the game. Decisions are made about the game scene, objects, ways of interaction etc. Changes are made to the previous specifications because new ideas that can improve the game experience for the player arise. For example, in this stage the designer of the game came up with the plan of inserting power-up boxes, that would aid the user by providing certain benefits.

The game scene contains a close to dusk skybox, a beautiful body of water, the player's vehicle, a finite number of enemies and a finite number of power-up boxes. There are two types of objects in the scene: dynamic – which means they change position or interact with other objects, such as the flying car, the enemies, the power-up boxes and the monolith, and static – which means that they do not interact specifically with any other objects and do not change attributes, like buildings and the body of water.

Table 1. Scenarios and actions of the game

Scenario	Actions
S1. Navigation in the 3D scene	T1. controlling the vehicle using the mouse movements T2. increase speed by pressing space T3. zoom in and out using the scroll wheel
S2. Attacking and avoiding enemies	T1. observing the enemies T2. flying towards enemy T3. player attacks by pressing the left button of the mouse T4. the enemies attack when the player gets in a certain range and in a certain field of view T5. observing the enemies reaction T6. avoiding enemies
S3. Monolith	T1. the player should understand the objective, by reading the message shown on the screen T2. successfully navigating in the scene T3. observe the monolith T4. fly towards objective T5. message of winning the game
S4. Repair power-up box	T1. recognizing the object T2. flight towards the objective T3. colission with the object T4. object destroyed T5. life health increased
S5. Immunity power-up box	T1. recognizing the object T2. flight towards the objective T3. colission with the object T4. object destroyed T5. enemy attack canceled for 20 seconds
S6. Display relevant messages	T1. message with the game objectives T2. toggle help option T3. quit button T4. player health information T5. message of collecting repair power-up box T6. message of collecting immunity power-up box T7. message of destroying enemy T8. message of losing the game T9. message of winning the game

Now that the gameplay has been defined in the previous stages, the main interaction techniques should be established. It is useful to create scenarios that are composed of specific actions. Each one of them represents a certain part of the game and they will be useful during the implementation and evaluation stages. Another change has taken place at this stage. Initially, the interaction between the player and its vehicle was established to be done using keyboard buttons W, A, S, D. But in an initial implementation prototype of the game, it has been seen that it is cumbersome to control the movements, so another option was chosen – using a mouse so that the vehicle will follow its position on screen. The scenarios and actions for EvoGlimpse can be seen in Table 1.

The car has a set of parameters that can change during the game, such as life amount, speed, attack damage, and fire rate. The player can change the car's position using the mouse in a continuous interaction mode, while the camera perspective can be changed with the scroll wheel (zoom in and out), and the attack action can be done by pressing the left mouse button. The enemies have similar parameters to the player's flying car, and additionally, they have AI (Artificial Intelligence) capabilities – because they need to move on their own, without exterior control.

The most important metaphor in the game is the player's interaction with the power-up boxes. By touching one of them with the car, some characteristics of the vehicle or of the enemies will change. The color denotes the class of the box, that is what attribute it will modify. Each box appears with a certain probability, and in the scene, at the same time, there will be a limited number of them. Some boxes have as a parameter a quantity which says with how much a certain attribute changes – it can be a fixed number or a random number from a certain range. For example, the 4th game scenario this will be relevant for the repair power-up box. Other boxes give the player abilities that expire after a few seconds. For example, in the case of the 5th game scenario, this would be the immunity power-up box.

5. Implementation

In this section, we will discuss about the actual steps of the game implementation. The tools that were used will be shown, together with the game objects, the game scene, and details about how each scenario was created will be given.

5.1. Tools

The first software tool that was used is Unity 2017.3.1 (Okita, 2014), which is a platform for creating both 2D and 3D games, which can be ported on different platforms or operating systems – Windows, Linux, Oculus Rift etc. The implementation is done in C# in files called scripts; these handle the object logic and the results of the interactions. Unity offers a lot of game development possibilities, such as maps, terrain, shadows, packages with premade particle effects etc. which make the process of creating a digital game simple, fast and easy.

For additional object modeling or to change premade objects, Blender 2.79a (Wartmann, 2000) was used, because it is a very popular and efficient graphics tool. To create the prototypes and edit textures for the game objects, Adobe Photoshop CS6 (Onstott, 2012) was used. Also, it is important to note that the whole development process was documented, and whenever there were updates or maintenance steps, these were recorded.

5.2. Game objects

To create an immersive game world, after defining the story and having in mind the inspiration sources, game objects were chosen to fit the desired goals. Some of the 3D models, objects and particle effects come from the Asset Store of Unity, some come from websites that make them available for free. Due to space considerations, they will not be displayed in here. Now that we have all the necessary game elements, it is time to create the interactions between them so that the game can finally be played!

5.3. Implementation details

The creation of the game scene composed of the water body, the skybox, and the buildings was done first. This has been done directly in the Unity development tool, in the scene part. Also, at this point in time, it was realized the need for a region delimiter, to help the player recognize better the area in which the game takes place, so it wouldn't wander too far away. After importing the buildings in the game scene, some components like rigid bodies and colliders with physics materials were added to them, so that the player's vehicle will collide with them and not pass through them. The result can be seen in Figure 9.

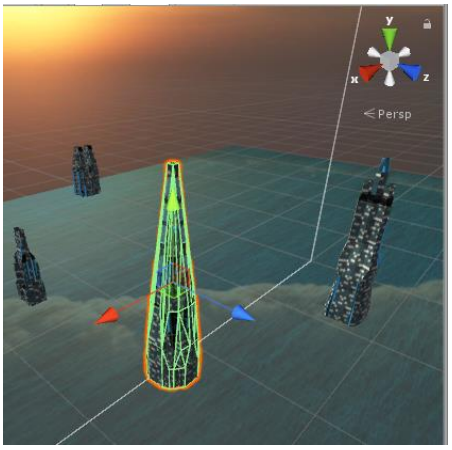


Figure 9. Game scene after importing some buildings and positioning them. The collision mesh attached to the selected building can be seen.

In the next step, the interactions between the player and the flying vehicle were implemented. The main camera was attached to the body of the vehicle, behind it, so that when the player moves, the camera will follow the movements and it will change its position. Two scripts were added to the player's object. The first script is Vehicle Pilot and it contains attributes such as speed, health, and attached game objects for certain text information messages. The Update method contains code for:

- starting the game when the Y key is pressed (which is required when the game starts, from the message "Start? Press Y….");
- stopping the game when the escape key is pressed, and which will quit the application;
- increasing the speed of the vehicle when the space key is pressed;
- zooming in and out when the scroll wheel is used; this actually changes the field of view of the camera;
- changing the position of the car by following the position of the mouse of the screen, and changing the rotation by accounting for the angle between the up axis of the object and the mouse position;
- smoothing the movement of the camera.

Three additional methods are created here, and they deal with the amount of life of the player. In the method TakeDamage, whenever an enemy successfully hits the vehicle, the amount of life is decreased with a certain given value. When the life quantity reaches zero, the method Die is called which means that the game is lost. The time is frozen, and a specific message is displayed on the screen. The third method is called when the

Figure 10. The object used for representing the player's flying vehicle. Two important elements can be seen: the camera attached to the back, and the particle effect attached to the front – which appears only when the fire action takes place.

player used the repair power-up box. It is called HealUp and its effect is that the player gains back the whole missing amount of life from the maximum that it can have.

The second script attached to the flying vehicle is used to deal with the interaction of the player with the attack techniques. The attack has attributes such as damage, range – the minimum distance required for the player to be able to actually hit something, fire rate or attack speed – how fast can the attack reload when the mouse key is pressed, and impact force. Also, it has attached a camera – which is needed to direct the attack, and a particle system – that appears when the player shoots. These can be seen in Figure 10. The Update method continuously checks if the Fire1 button has been pressed – this is the left button of the mouse. If so, the Shoot method is called after a reload time that is the inverse of the fire rate. In this method, the particle effect is played, and a sphere cast with radius 4 is used to perform the actual attack. If the ray hits an object it displays bullet effects on its surface, and it checks that its type is Target. If so, it means that we are hitting an enemy, so we can subtract a certain amount from its life. To aid the player, a crosshair is added as a circle in the middle of the screen. This means that the attacks will hit in the center of the screen, so the players will know how to position the car accordingly.

Now that we have a functional vehicle that can attack, we need enemies. But we don't want dumb enemies, we want ones that are able to move and react to our presence around them. To do this, we need to add AI capabilities, which for video games means creating a state machine. For what we want, there need to be a total number of 5 states:

1) Initial – the enemy is initialized at a random position in the game world;
2) Idle – the enemy checks the environment continuously, by rotating in a circle, around a pivot, but does nothing else;
3) Fly – if the player is at a certain distance smaller than a set value, and if it is in its field of view (for example, 60 degrees), the enemy flies towards the spotted vehicle;
4) Attack – and attacks;
5) Die – if the amount of life reaches zero, the object is destroyed.

These states and the transitions can be seen in Figure 11. The implementation is straightforward, because the necessary conditions are checked, and decisions are being made depending on the position of the player. The implementation of the movement is done by using rotations around a pivot point, with a certain speed, and around the y-axis. The implementation of the attack is similar to the player's attack. In order to make the enemy "see" the player's vehicle, we have to compute the distance between these two, and also the angle between their positions. If the distance and angle conditions are met, using the quaternion function Slerp, the enemy changes its heading towards the player in a natural motion movement, and it changes its position forward. If the enemy is close enough to the player, it starts attacking. The player can outrun the enemies and escape their attacks.

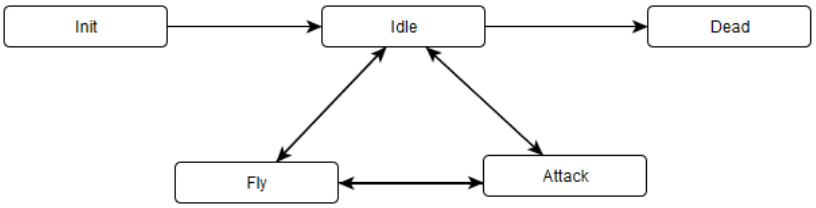


Figure 11. State machine for enemies

The enemies have two scripts attached to them: one for chasing and attacking the player – which contains the information previously discussed, and one for rendering itself as a target. The latter is used by the player's vehicle to check if it attacks the right objects. The enemies themselves have a certain amount of life too, thus they can take damage when hit by the player's attacks. When the health amount reaches zero, they are destroyed, an explosion particle effect plays, and they are removed from the game scene, and a specific message is displayed as seen in Figure 12.

Figure 12. Particle effect and message that appear when an enemy is destroyed by the player

The player might need help to survive longer in the game, so power-up boxes were added. There are only two of them for now. The first one is called repair, and it has a green wrench on it, to help the user recognize its meaning during the game. Its purpose is to give back to the player the lost amount of life. It is important to note here, that this happens when the player's vehicle collides with the box. To make it easier for this to happen, a collision box is added, but this expands a little bit outside so that if the player misses is by little, it will still have an effect. On collision enter, the function HealUp previously mentioned when we talked about the flying car's scripts, is called, the repair box is removed, and a message is displayed on the screen to announce the player that the changes have taken place.

The same is true for the immunity power-up box. This is represented by a rectangle that has a blue shield on top of it, to help the user recognize its meaning during gameplay. The ability that is rendered to the player lasts only 20 seconds. This means that the flying vehicle gains a shield that cancels all enemies' attacks. This is done by bringing their attack damage to zero. Now the player can go near the enemies, they will still follow him, but their attacks have no effect on the player. After the 20 seconds have elapsed, the function that gives back the damage to the enemies is called.

Another thing to discuss is about the displayed messages that appear on the user interface. There are 3 types of messages: static ones – which do not change during the game, like the help menu, or game instructions, dynamic ones – they change depending on the game status, for example, the amount of life of the player, and triggered ones – which, as their name suggests, are triggered by specific interactions, like destroying an enemy, or collection a power-up. This was not so straightforward to implement, because at first each message was attached to the object it was related to as a text object.

This proved to be incorrect since we wanted to make the trigger messages pop up for a few seconds and then disappear. This meant that we had to destroy them, but when we tried to display them again for the next event, they were gone. So, the solution was to create a separate script called HUD_Manager, which manages the head-up display, that is the information relayed to the user. In here we linked all text messages as game objects, and set them inactive by default. We created a function that sets one such game object active, then waits for a few seconds, and then renders it inactive again. For each triggered message, we called this function, which proved to be the right way of doing it.

By now, you are probably wondering how the game can be won. We have talked in the early development stages about the monolith, which is the key of the game. This object spawns at a location far away from the player. It is difficult to spot it right away, which makes the game more exciting. But if the player finds it and flies towards it, when it gets close enough to it the game is won. This is done by checking the distance between the player and the monolith. The downside is that its position is fixed, but this option was chosen because the game scene is small.

6. Evaluation

For implementing and running the game, we used the following hardware specifications: Intel(R) Core(TM) i7-6700HQ CPU, 2.60GHz, 8.0GB RAM, 1TB memory, NVIDIA GeForce GTX 960M, on Windows 10 operating system.

The game was evaluated in two stages. First, the designer tested the functionality, after each step of the implementation. This was done to check the correctness of the behaviors. When errors were noticed, solutions were found and implemented. The second stage consisted of a heuristic evaluation. Nielsen's 10 usability heuristics for user interface design (Nielsen, 1993) were used. These are: the visibility of system status, match between system and the real world, user control and freedom, consistency and standards, error prevention, recognition rather than recall, flexibility and efficiency of use, aesthetic and minimalist design, help users recognize, diagnose, and recover from errors, and help and documentation. Details about the requirements for each of these can be seen in Table 2. Two evaluators had to independently complete reports for each game scenario, consisting of marks from 0 to 100 for each usability criteria, and they highlighted the errors that they found. Then a group evaluation was done by the two evaluators and the game designer, where the game

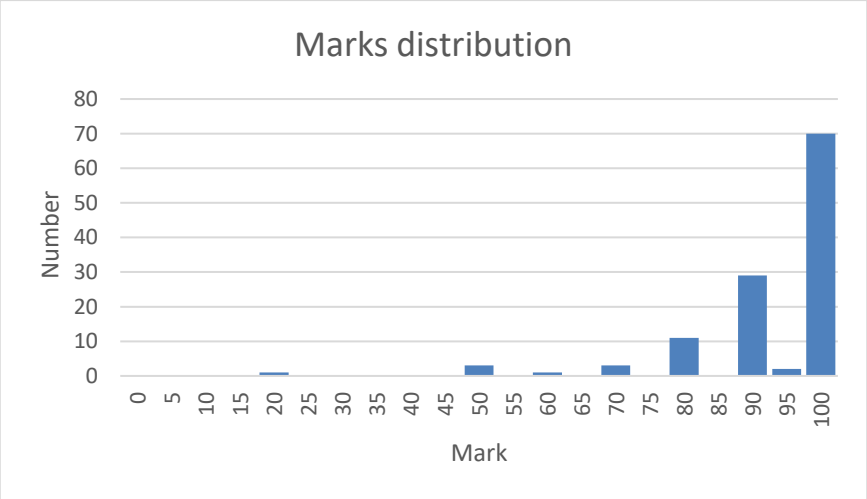
Table 2. Evaluation criteria

Nb.	Questions and requirements
1.	Visibility of system status • Is the state of the system visible at all times? • Is the feedback offered by the system suitable? • Is the response time appropriate, without unacceptable delays?
2.	Match between system and the real world • Does the game correspond to the mental model that the user has from a real-world game? Is it what you expected or similar to other games? • Are the language, words, and phrases used familiar to the user? • Is there a natural way in displaying the information? • Is this a suitable shooter game? Is the game scene realistic?
3.	User control and freedom • Can the user execute the necessary actions to fulfill the scenarios? Is their functioning correct? • Can the user exit an unwanted state? For example, is there a need for an undo/ redo button? • How does the vehicle control, attack, collection, and buttons feel?
4.	Consistency and standards • Is the user surprised by different words, situations or actions that have the same meaning? • Is there consistency in the use of colors and symbols? • Is the meaning of the objects from the scene understood?
5.	Error prevention • What is the functional correctness level of the game? • Are the errors eliminated or are there methods to prevent situations that favor the apparition of errors? • For example, notice what happens if the player tries to get too close to the water, at the collision with different objects etc.
6.	Recognition rather than recall • Can the player recognize the objects and their usage? • Are there elements that require storage in the memory of the user?
7.	Flexibility and efficiency of use • What is the level of flexibility and efficiency of the game usage? • Is the user bothered by certain aspects? Or are some of them missing?
8.	Aesthetic and minimalist design • What is the quantity of relevant information? • Is there any redundant information? • Is the information presented clear and easily accessible? • Is the field of view of the player cluttered with too many elements or is it suitable?
9.	Help users recognize, diagnose, and recover from errors • Are the messages clear and helpful for the player? • Should there be any additional error prevention cases?
10.	Help and documentation • Is the help menu complete? • Does it contain clear, simple, and easily accessible information? • Is the documentation clear, does it contain sufficient information for the player? If not, what should be added?

was tested once again, and the problems were pointed out. The game developer came with solutions for the flaws that were discovered, and this process proved to be very easy and efficient. Also, the game obtained a 92.6% usability rate, which is quite satisfying.

The main functionality flaws had to do with the collision of the car with the buildings – which can be solved by altering the bounce parameter of the physics materials. Also, the users are not fully satisfied with the level of entertainment of the game, but this can be changed by making the game scene bigger, adding more enemies, with more variety to their behavior. The marks that were obtained for all six scenarios and for the 10 usability heuristics are summarized in Table 3.

Table 3. The number of marks that were obtained during the heuristic evaluation



For each scenario, a series of solutions were proposed by the developer, as follows:

- **Scenario 1:** There is a problem at the level of materials that are attached to the objects, in particular to the vehicle and the buildings. This can be solved by changing the bounce value in the phycis property of the materials. The creation of a menu will be taken into consideration for the next implementation iteration.
- **Scenario 2:** The enemies have attack particle effects, but those can not be observed since they are behind the player. This can be changed by adding effects on the car, and adding sounds that would help the player know if he / she is shot. Also, the user attacks in the center of the screen, where the crosshair is

displayed. The player should experiment with the attacks, and thus it can be seen that the enemies can be shot only at a certain distance. The enemies do not die instantly, as it can be seen on the particles displayed on the player's vehicle, multiple shots are needed. A health bar should be added to the enemies to aid in this problem. Also, the enemies only attack if the user is at a certain distance from them, and in a certain field of view. To make their response faster, their movement speed can be increased.

- **Scenario 3:** The option of fixing the position of the monolith was chosen because the game scene is small and the objective would have been too easy to find. If the scene was bigger, then yes, the position of the monolith would be randomly computed at each run of the game. The same is available for the enemies and the power-up boxes - if the game scene is bigger, more objects can be inserted, thus making the game more entertaining.
- **Scenario 4:** The option of making the boxes dissapear after the collision was chosen because it would have been confusing otherwise. It can be noticed that they do dissapear instantly after we touch them, and the interaction with them is correct since their effect is immediately observed and a feedback in the form of a system message is displayed. Their necessity can be increased by adding different abilites to the enemies or making them smarter.
- **Scenario 5:** The focus was more on the functional correctness of the game rather than on the level of entertainment. This can be changed by increasing the game scene and adding variety to the enemies.
- **Scenario 6:** The best solution is to do error prevention at the collision with the buildings, so the user won't have to worry about it.

As future improvements for the digital game developed in this paper, we can think of making it multiplayer. This means that more than one person can play the game at the same time. This would make the game more difficult, but also more entertaining. It was also noted that this type of game would be suitable for smartphones, because it would be easy to control the movement of the flying vehicle with the change in position and orientation of the phone, and by tapping actions to shoot and to speed up.

7. Conclusions

This paper focused on presenting the development methodology and evaluation of an interactive application. The purpose was to implement specific development stages to build a video game. Literature was reviewed in order to understand the necessary steps. Then, detailed explanations were made at each stage. We explained where we got our inspiration from, and how we planned the game. Then prototypes were created to design the user interface of the game. Afterward, the game scenarios were established. Following these, the game was implemented in the presented technologies. We showed what game objects were used, and how each one of those was inserted into the game.

In the last stage of development, the functionality and usability of the game were evaluated. It has been found that the game has a high level of usability. This evaluation represents a big help for a creator of interactive application because it is an extremely helpful way of finding in a fast and efficient way what the problems are, which speeds up the process of improving the system. This whole process proved to be a lot of hard work, and documentation was needed to keep track of the update and maintenance steps.

Acknowledgement

The authors would like to give thanks to Selma Evelyn Cătălina Goga and Al-Doori Rami Wathek Yaseen for the help they provided during the evaluation stages of the game.

References

Adams, E. *Fundamentals of Game Design*. New Riders Publishing, 2009.

Barnhardt, P. Game design tips 2: That Pesky Character development. Vol. AXS Digital Group LLC, 2011.

Bethke, E. *Game Development and Production*. Wordware Game Developer's Library, Wordware Publishing, 2003.

Clarke, A. C., Kubrick, S. *2001: a space odyssey*. 1968.

Dawkins, R. The Greatest Show on Earth: The Evidence for Evolution. Free Press, Transworld, 2009.

Enix, S. *Life Is Strange*, 2015.

Mitchell, B. L. *Game Design Essentials*. 2012.

Nielsen, J. *Usability Engineering*. Morgan Kaufmann Publishers Inc., 1993.

Nolan, C., Thomas, E. *Dunkirk*. 2017.

Okita, A. Learning C# Programming with Unity 3D. A. K. Peters, Ltd., 2014.

Onstott, S. *Adobe Photoshop CS6 Essentials*. SYBEX Inc., 2012.

Pinelle, D., Wong, N., Stach, T. *Using genres to customize usability evaluations of video games*. Proceedings of the 2008 Conference on Future Play, 2008.

Salen, K., Zimmerman, E. *Rules of Play: Game Design Fundamentals*. The MIT Press, 2003.

Sellers, M. Designing the Experience of Interactive Play. Playing video games: Motives, responses, consequences. Vol. Vorderer, P. & Bryant, J. Mahwah: Lawrence Erlbaum Associates, 2005.

Serdar, A. Digital Educational Games: Methodologies for Development and Software Quality. Dissertation. 2016.

Villeneuve, D., Fencher, H., Green, M. *Blade Runner 2049*. Warner Bros. Pictures, Sony Pictures Releasing, 2017.

Wartmann, C. The Blender Book: Free 3d Graphics Software for the Web and Video with Cdrom. No Starch Press, 2000.