University of Europe for Applied Sciences

Bachelor of Game Design

Ousama Andari

HOW TO MANIPULATE A GAME ENGINE AND LEVEL DESIGN TO CREATE GAMES WITH LESS CODING SKILLS?

BACHEOLOR'S THESIS

FIELD OF STUDY: GAME DESIGN

Matriculation Number: 29757500

7th semester – winter 2021

FIRST ADVISOR: PROF. FLORIAN BERGER

SECOND ADVISOR: CHRISTPHOR ETMER

SEPTEMBER 2021

Contents

Abstract

The process of creating a game is determined by complex procedures which game designers encounter. One of the central issues discussed in the modern game industry is focused on the possibilities to create a game with minimal coding skills. This study aims to determine the various ways of using techniques that are valuable alternatives to complex coding mechanisms. In particular the work focuses on the creative key options in game design that overcome programming: manipulating game engine tools, game mechanics, level design, visual scripting and the resting game engine elements. The results showed a noticeable advantage in the game industry where it is possible to create modern games with less coding knowledge. In such a way it brings game design to a new level where almost every person can create a game with less coding problems. This in turn reduces the amount of obstacles to make a game and allows to find innovative methods in game design.

Keywords: game design, game engine, coding, level design, game mechanics, visual scripting

1. Introduction

A lot of game designers struggle through tons of issues that might stop them from any further process in their game project, one of the biggest and common challenges is the lack of coding skills that can lead to complications and lots of errors in a project. However, we believe there is always a solution for every problem, in most cases, it would be solved through coding, but sometimes developers find a way around and use whatever the game engine provides from different tools, features, assets, scripts, packages, etc. From what was experienced in the sector of Unity Engine, the majority of times the arising problems were concentrated around coding a specific mechanical or movement that would affect the game. Usually, Game Engines are big enough to explore and mess with, for example, there is a huge store designed especially for unity engine to help developers save time and effort and maybe help them solve some of their problems. In addition, there are many tools and packages that could probably help fix some issues.

"There is no one great theory of game design because every design decision has many different consequences" (Sylvester,2013).

For example, some game developers might decide to make a game with a water slide in it, but they probably did not know how to code it, and instead, they used some physics material that game engines provide, to decrease the friction between the object and the slide and they ended up having a perfect slide mechanic that was made with no code. So whatever happens behind the scenes remains behind, and all we need is the functionality of the game itself and if people end up enjoying it or not, regardless of whether it was coded or done in some other creative way. Imagine getting stuck in the middle of developing a game, because of a programming error or having difficulties coding a specific game mechanic, it would be very frustrating and disappointing not to find a way out and finish the project. It might be that the code was more complicated than initially thought, in this case, it is necessary to think about other ways that can fix the issue. For instance, in case there is no possibility to make the player teleport from a specific location to another one. Instead, there is a creative way of making a clone of the current scene and triggering a change scene event with a new player spawn location. Once the player collides with the teleportation door, in the next scene the player is going to spawn in wherever location he is set to. There is always a possibility to test it and see if it works, however, it could cause other issues that are not related to coding, so being familiar with the Engine that is used

it is possible to achieve the main goal. Sometimes it will not be the most efficient solution, but most importantly is the ability to fix the issue. Even if it took more time and effort, it will still be able to get through either with the implementation of some basic coding or without it at all.

To explore this topic, a range of **research questions** were established in the paper. The key points, in general words, could be summarized as follows:

- Is there a possibility to create a game without coding?
- Is there a specific design theory to make a successful game?
- Can game engine tools be used differently from their initial purpose?

2. Main part

2.1 Game Engines

“So what are game engines? In today’s modularly constructed games, the game engine refers to that collection of modules of simulation code that do not directly specify the game’s behavior (game logic) or game’s environment (level data). The engine includes modules handling input, output (3D rendering, 2D drawing, sound), and generic physics/dynamics for game worlds” (Lewis & Jacobson, 2002). “So far it seems clear that a game engine is a piece of software that enables the creation of video games. However, we still do not have a proper definition of what can be expected in such frameworks. The main goal of a game engine is to abstract common video game features allowing for code and game asset reuse in different games”(Andrade,2015). There are many Game Engines that can be used to develop a game, however, they might differ in a way or another, but most importantly that they are being used for the same purpose which is making a game. Some game engines were made to focus on making 3D games more than 2D, and some engines provide more tools or features to make 2D games, and so on. It all depends on what type of game you want to develop and what specific tools or graphics you want the game to work on. It is a matter of choice and it will sometimes depend on someone’s skill or knowledge in that game engine. One of the most popular game engines nowadays is unity and unreal, and the main reason why they both got popular is that they are free to use and provide enough tools and access to a huge assets store where developers can get free or paid assets to use in their game projects. Not to mention that both engine graphics are stunning in both 2D and 3D.

2.1.1 Game Engine Tools

A game development tool is a special technology that allows developers to use it in a way to add specific game development requirements such as animation, sounds, user interface design, and more. Most of the engine tools are designed for a specific use or action but sometimes it is possible to use a tool creatively to add something new or unique to a game in an unusual way. There are no strict rules on how we could use a tool, so we do not have to use them only in the way they were intended for. Few common tools are frequently used by game developers in unity. Some of the popular tools are the animation graph and timeline, physics materials, tile map editor, and builder pro. These tools can be used in creative ways to come up with Incredible and ideas or solutions.

2.2 Designing games

“People trying to understand games often compare entire games to each other and try to pick out how their differences affect play. But it is hard to find design principles this way. There are so many differences between the games that it’s impossible to isolate the effects of any single one” (Sylvester, 2013). There are two popular games that both were made in the same game engine by two different companies, first one is Hollow knight and the second Is Cuphead. It is almost impossible for players to see any differences in both game's design principles, For example, Hollow knight used a lot of visual scripting whereas Cuphead relied on coding and a very unique art style. Both are great games and run smoothly with no problems. Developing games is not always attached to only complex programming skills but it rather includes the ability to use our knowledge that allows to implement them in a creative manner to get through most of the technical obstacles. In this framework, some of the technical issues which might appear are a lack of programming and knowledge in a game engine.

“The designer watches three people play a game, and observes the patterns in their experiences. He sees them understand certain things but not others, take certain actions, remember certain events. Then the designer changes one variable in the design. With the next three testers, he watches the experience change in some specific way—one new idea understood or lost, one memory gained or erased. The designer learns that their one small change caused that one characteristic effect” (Sylvester, 2013).

Every little decision developers make on their games ends up changing one or many different gameplay elements in the game, for example changing horror game sounds and music to some other sounds that do not suit the gameplay atmosphere. This will give players the wrong gaming experience that they were supposed to have. On the other hand, if a developer decides to change the speed of enemies in the game from normal to fast speed then the whole gameplay has changed and probably became difficult than before. Every design decision has it is own implication on the project and if developers end up deciding on something then they will bear the consequences.

2.3 Game mechanics

Behind every successful game there is a good mechanic, and making a good mechanic isn’t an easy task, especially since most developers lack coding experiences which makes it much harder for them to implement it. “Every game mechanic has a price tag. It costs design effort since it must be implemented, tuned, and tested. It costs computing resources which we could have used somewhere else. It might force changes in the fiction, or blur the focus of the game’s

marketing. Players submit themselves to these costs because they want a meaningful experience. Good design means maximizing the emotional power and variety of play experiences while minimizing players' comprehension burden and developer effort. This form of efficiency is called elegance" (Sylvester, 2013). It might be frustrating to implement a game mechanic due to the lack of coding experience and therefore some developers resort to another solution such as visual scripting or other creative fixes. Players want a good gaming experience from games, most of them do not care about how the game was made. There are plenty of game engines and ways to develop games nowadays, some games were made in engines that do not require any coding at all, but players still liked those games and enjoyed playing them. Coded game mechanics might be much more convenient because with code programmers can be very detailed and specify the game mechanic in any way they want. This will make a game mechanic function perfectly, however, it is necessary to take into consideration that a lot of errors might occur, which will make developers consume more time and effort and sometimes with no good results. Sometimes coming up with a simple mechanic is the key to a successful game, however, a simple mechanic could be hard to implement because it normally requires a very unique design that aims to support it in a way. Making a game with a simple mechanic means a lot of testing, unique designs, and effort. "Flappy Bird" is an example of that, as it has only 1 simple jump mechanic with a wonderful design**.** However, as a developer, you have to think about the design of the game you are making and if this simple mechanic is going to fit the design or not. "Simple mechanics smell like elegance. Elegance is as much about reducing the cost of a mechanic as it is about increasing the benefit. A bloated, overly complex mechanic might create good results, but it's often not worth the learning burden it puts on players. Simplifying it might mean losing a few nuances in play, but it also opens up Mindspace for other, more efficient designs. Furthermore, reducing the complexity players are exposed to increases their appreciation of what's left. Players who aren't overburdened will fully explore a game and enjoy every morsel of experience"(Sylvester, 2013). In addition, most of the simple mechanics are being used in multiple ways, an example of that is a jump mechanic, it is simple and it has been used in a lot of different ways with unique designs to create new gameplay experiences. Some games have used the jump mechanic to jump from one platform to another, whereas other games like "Flappy Bird" have used it to control the player in a vertical movement, it is a slight difference but it required a huge change in design. As stated by (Sylvester, 2013), a mechanic that can be used creatively, effectively, strategically, and tactically is much more wonderful than one that can be used once and only fill a single hole. Not to mention that it can be a unique tool that allows to create of new connections and a variety

of possibilities that intern play different roles. “This far it is more than clear the functional importance of game mechanics. But, where does the fun come from? What makes players like or dislike a game mechanic? To start with, every game mechanics is characterized by its semantics, which will inevitably determine the first-impact appeal, making the player like or dislike it. In other words, some persons, due to their cognitive backgrounds, might be attracted by the idea of playing with a ball as soon as they see one, and others might not. Game designers can do very little about it, besides choosing mechanics whose semantics seem to be appealing to the target players, and ensuring that they are coherent with the context and the goals of the game”(Fabricatore, 2007). To make a good game we might not only need good mechanics but the right mechanics and design for the group we are targeting, if we decide to have a shooting mechanic but our target group is children of 13 and above then we must make an appealing design that suits this specific group to get the best results possible and ensure safe gameplay for the children.

2.4 Manipulating A Game Engine

Manipulating a game engine might sound complicated but we always have several things to mess around inside of a game engine, although sometimes it might lead to errors and crashes if it is overdone. There are many built-in tools and premade assets that help manipulate an engine. Therefore there are too many creative possibilities to come up with and change the use of something in a different way. Every tool or object is meant to be used for a specific goal, but sometimes developers find creative ways to use these tools and make something different or unique out of them. Game engines are very complicated and if we want to understand them we have to invest enough time learning the tools and technologies it provides. Manipulating a game engine does not sound clear enough, in fact manipulating a game engine means manipulating everything inside of it from tools, features, technologies, physics, etc.

2.4.1 Manipulating Game Engine Tools

All game engines have tons of tools and plenty of features and possibilities to come up with, so if we take unity as an example, in a default layout you will find the animation and many other default tools, game developers normally use a tool to animate characters and bring them to life, instead of having a moving box. But what if this tool was used somehow else? Let’s say there is a 2D game with a player in it that already has a movement script and all we are missing

is some platforms where the player can move or jump on. Adding platforms is easy but adding moving platforms could be tricky, so since we can animate objects in game engines such as unity, then we can try to animate a platform to make it go up and down and make its animation in a loop. In unity after making any animation, there is an option if you want this animation to keep going in a loop, and in this case, if we have a thin rectangle and we want to move it up and down continuously we can either do it through code if we had enough experience. But because coding is not that easy to understand or learn, we can try using some other possible options such as the animation tool to make objects in our games move, rotate, and scale up or down, etc. Another example would be using the same tool to try to add any type of enemies or obstacles such as the missiles in the "Jetpack Joyride" game that was made by "Half Brick studios". In the game, there is a part where players get attacked by random missiles, although those missiles might have been done through code, there is still a possibility that they could be animated to appear at a specific time after we add a collider that triggers the player collision once he hit a missile, which makes the game end.

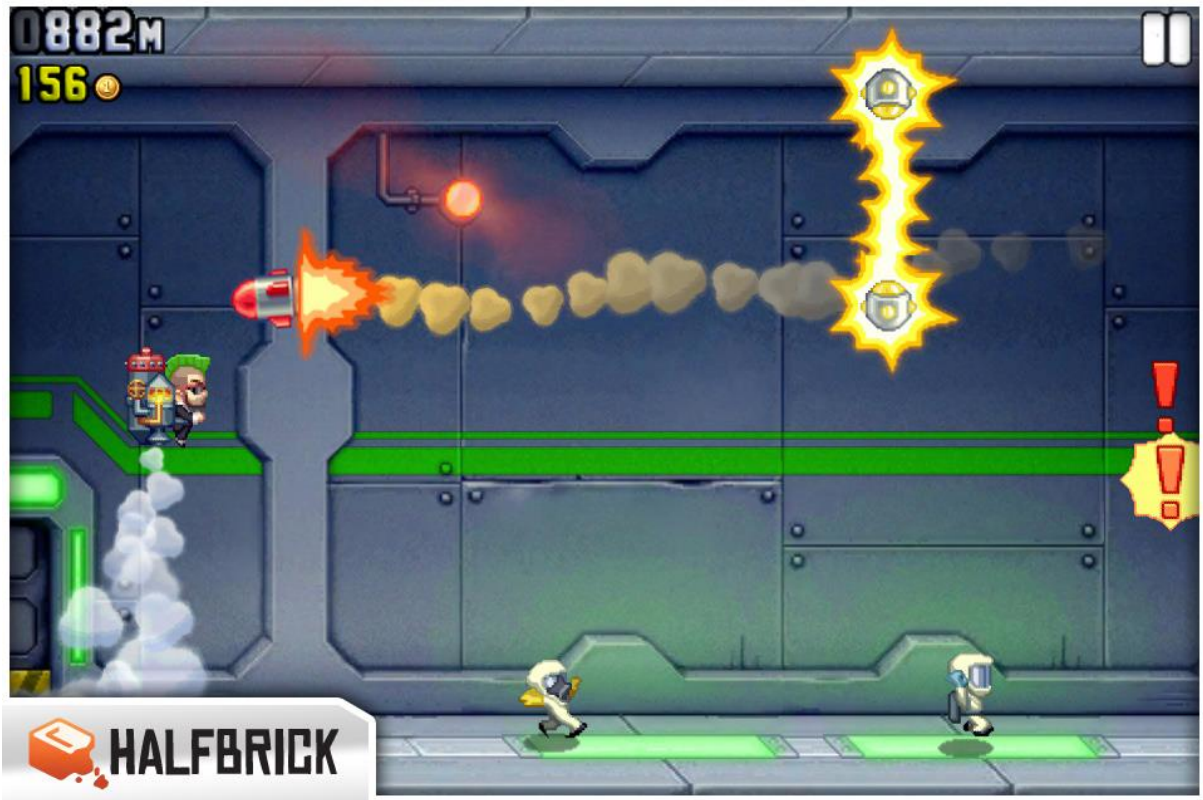

Figure 1 Example of animated missiles (https://www.dualshockers.com/jetpack-joyride-receives-another-free-update-lots-of-goodies-included/)

There are too many other tools and similar examples that show how can a tool be manipulated or used creatively.

In unity there are also preprogrammed objects that can be used in many ways such as User Interface (UI) objects which are mostly preprogrammed to react once added to a scene. There are objects that unity will detect or recognize as buttons and whenever a mouse hover over it, it will interact as a clickable button. Buttons in unity will not work automatically, but it already interacts with the mouse or any mobile screen as a clickable object. In addition all buttons in unity can be customized: a text, image, color and cool designs can be added to a button to change the aesthetics of it or to be used somehow else. Manipulating buttons would be possible if we were making a game similar to “Where is Waldo” and we wanted to find a small character in a huge image with many details in it. Making a scene with many characters in it is easy because all you need to do is draw many characters and this does not require any drawing skills but time. The goal is to find a specific character among all of the other details in that image, that can be coded if you already have the knowledge to do it. However, the simple way to do it would be adding a button which is already clickable, then changing the button settings to be invisible on top of the character that need to be found, then adding a small line of code to switch the current scene to the next one once the player click on that character that has invisible intractable button. It is a very easy and fast way to do it but it can also have its own consequences and issues. In general having an issue that can be fixed through design is much easier than having an issue that can be fixed through code only. In the picture below the red

arrow is pointing at the invisible button that it is on top of the object that need to found, and on the top right corner there is a scaled up image showing the hidden object.

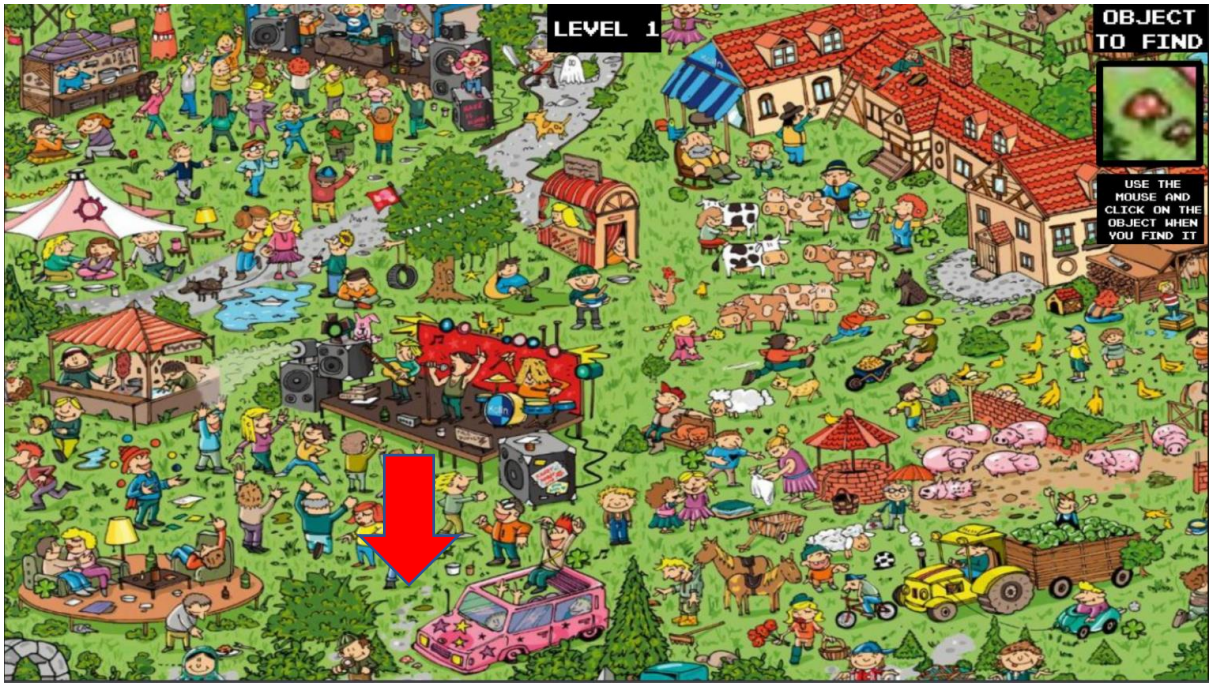

Figure 2Example of the invisible button (self-made game)

Moreover, if we made a game and added a boss to it, it would be a bit complicated to program that enemy boss to do specific actions depending on how close is the player to him. If the boss was animated, we can simply animate another object in his hand such as a weapon or some crazy effects and it all just goes in a loop without the need to code anything, although it is a bit complex to animate everything in this way for a simple boss fight that would be ideal skip to avoid programing, This picture is from my previous work, where the boss was fully animated to follow the player with no any animated weapons or effects and had only an edge collider that kills the player when he touches it.

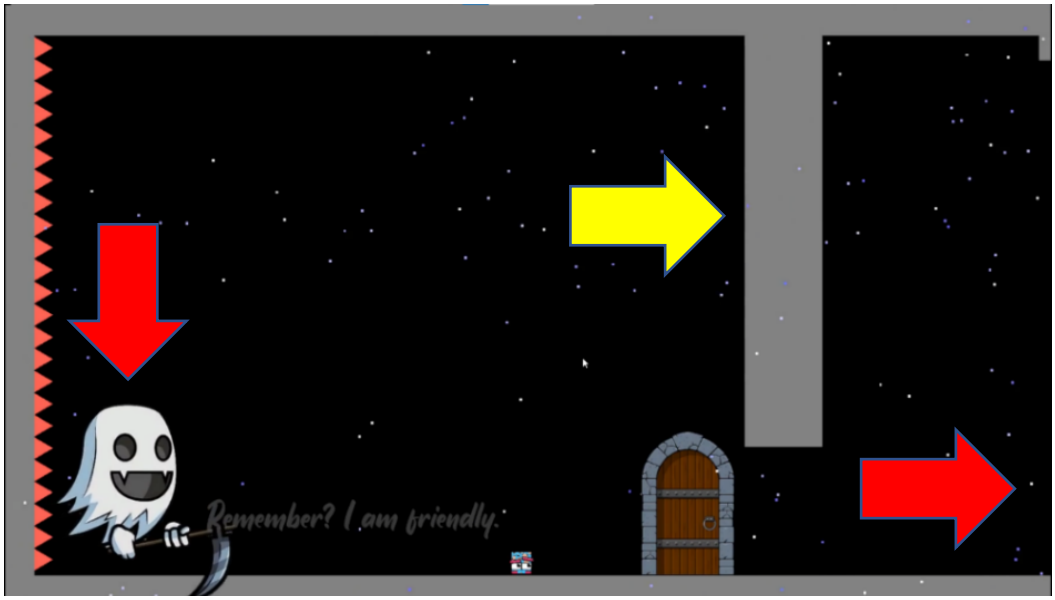

Figure 3 Example of using animation to add a boss that follows player (self-designed game)

The red arrows are pointing at the boss and the direction that it was animated to, whereas the yellow arrow is pointing at the animated door that opens after a certain amount of time to block the player until the boss wakes up, once the scene starts, the boss animation will play, the boss scales up and the door opens quickly and the run starts. It is great that there was a way out from coding all this, otherwise, it would have been painful to set on the computer trying to make a boss follow the player for hours. And for sure this might not be the most efficient way to make this type of gameplay and as long as it works without any issues it is completely fine to use it in any other situation.

2.4.2 Manipulating Game Mechanics

When we make games we first think about what kind of mechanics we want to implement and sometimes we get lost because of either the lack of ideas or having issues with coding the mechanic. But we can always find a way out if we try to be creative. First, we think about a certain mechanic that we want in our game, and after that, we try to implement it by code if there were enough sources or just try any accessible tool in the engine if that helps. If we fail to code it, then what if we make a special design that forces the mechanic to be automatically implemented? Or change the mechanic by adding a small twist to it? We can have a slightly new mechanic and be able to finish it. For example, if we have a game of 2 players and both players are asked to cooperate to reach a specific platform and proceed to the next level, let's say we could not make both of them jump on each other to reach higher platforms, in this case

what we could do is add a specific design that does the same job in a slightly different way, if there are boxes in the game and both players can push those boxes, then player A can push a box while player B wait below it so player A can get on top of both the box and player B to stay high and reach higher platforms without the need to jump at all, basically we made both players be able to reach higher platforms by adding some boxes to the game.

In the picture below player A is red and player B is green, both have to reach below to the right empty platform, to do that they are forced to push all these boxes which are going to fall and block their path, but to solve this puzzle player A and B are supposed to push boxes in some order to not block their path and to access the higher platform.

Figure 4 Example of using a creative way to replace a jump mechanic (Self-designed game)

2.5 Level Design

Most successful games are known for their good level design, mechanics, or narrative, but it is very important to understand the meaning of level design to get an idea of why it could make a game successful or not. Designing a level can be tricky and misleading sometimes so level designers are always very aware of any mistakes when they design levels. Level design is the designed environment that players explore and spend their time on, to reach the final goal and finish a game, in another definition level design is the process of design that is responsible for creating game levels or environments which are meant to lead players to reach the goal of the game. Unfortunately, some games lack good level design which is why most of the times players get lost or confused and start to get bored or just quits the game. Hence, the level design is important but isn't an easy task.

“The best level designers are never afraid to step back and re-evaluate their content. Often this requires a period of respite from the work in question; distance can clear up a clouded mind. A great designer isn’t afraid to throw content out or re-work a concept that needs attention. It is also extremely important for a level designer to recognize when he is becoming tired of his own work and when his work is not coming together. There is a huge difference between the two; in one instance a designer becomes weary of playing his own content over and over and is just sick of it” (Bleszinski, 2000).

2.6 Manipulating level design to avoid programming

Some game mechanics require a specific level design to give players the full experience and let them use the mechanic more to help them get a good feeling of it and learn it faster, so implanting a jumping mechanic without a platform might be boring whereas designing a huge tower with more platforms on it sounds more interesting. Manipulating level design can help avoid a lot of unnecessary coding, this can happen if we make a design that forces the player to do specific actions. For example, if we look at the picture below let us say we ended up designing a level that has three open doors, door A(red), B(yellow), and C(blue) but at the same time we do not want the player to get into door B, and we do not want to block door B in

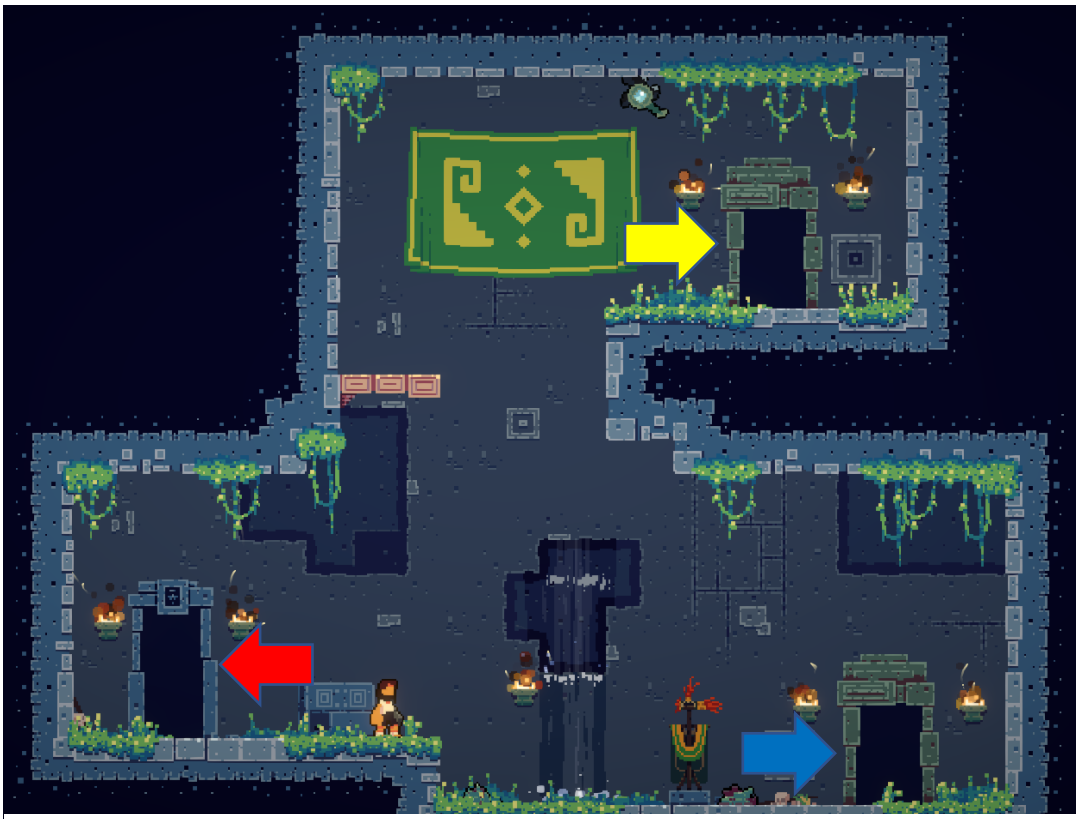

Figure 5 Example of manipulating the level design (Self-design game)

anyway because we are going to need it for later, we tried to block door B by code so it only opens at a specific time but we failed and the only option we have now is to design the level in a way where players won't be able to reach door B because it is high enough from them.

In this way, we have avoided coding to block a path, and instead, we changed the level design in a way that forces or limits the player to one or two options only. We can also try using animation and design together to make a specific action function such as block a door completely after some time, this is done by animating a platform to move in a specific direction and block a path after some time. It can be very frustrating to do it through code but another option would be to make it through a specific design and a perfectly timed animation. Another example is from the unique level design inspired by the game "Jump King" where the levels have tilted platforms to force players to lose a specific amount of progress once they land on them, in such a manner they will keep sliding with no stop once they try to jump to a platform and land on a tilted platform instead, therefore players have to avoid those tilted platforms or otherwise they get forced in each level design to keep falling to a specific point. The picture below shows how the tilted platforms were designed to force the player to fall in a specific path and lose progress.

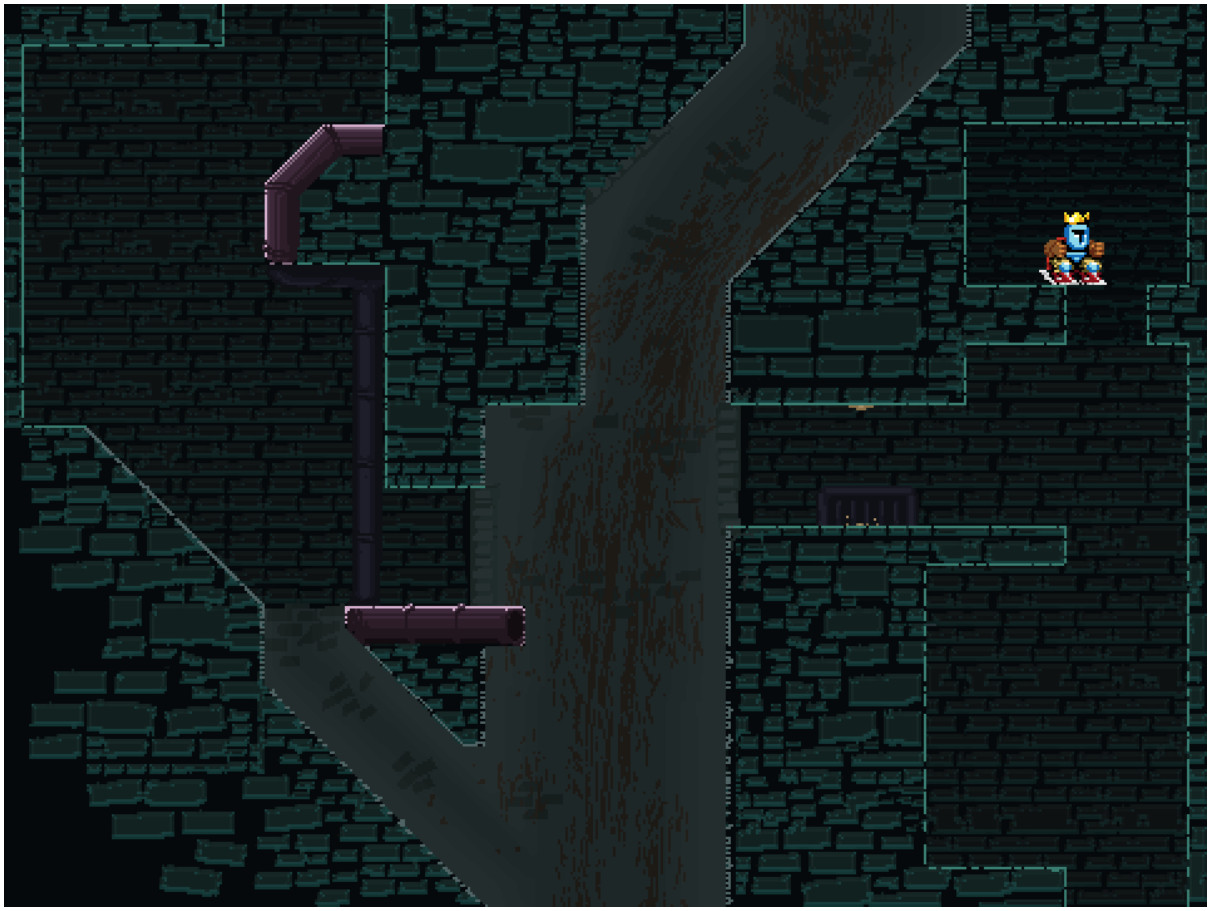

Figure 3Example of tilted platforms (https://steamah.com/jump-king-all-achievements-guide/)

In this game, the tilted platform was used to make players lose progress, instead of just making them die and reset from the last checkpoint. If we had a similar game but with a slight twist, for example, we would not want players to lose progress, but change their whole direction to a harder or longer path as a punishment for their failure. We could do the same design with less tilted platforms and once players make a mistake or fall on them, the tilted platforms will make them fall or lead them until they end up in a harder or longer path that will lead to the same ending goal but with more danger. In this way, we have forced players to follow specific paths and get punished depending on their own mistakes they have done in the game. Coming up with a smart design like this makes the game much more interesting and challenging but can also be annoying and frustrating for some other players. However, this makes the game unique in a way and excites people to play it or give it a try. Manipulating level design can also go wrong if designers are not aware of what they do, it requires a lot of testing and feedback to understand if the design is bad, annoying, misleading, boring, or impossible for players to get through it, any small mistake in design could make a huge impact on the gameplay. So it is very important to make sure the design is fair enough for players and without any single error. We can always come up with creative designs to force players to go through a specific path or

do a certain action instead of coding a feature that does it, but it will not always be the best idea or solution, because in some situations we got no choices but to make it happen through code and if we do not manage to do it, we can change the whole design which will for sure be a huge change and will add a twist and impact to the whole gameplay that could or was intended to happen at the first beginning.

2.7 Basic coding skills to overcome complex situations

No matter how often we try to avoid coding, it is still the main structure of any game and it is essential no matter the size of a game. Therefore avoiding it completely would be nearly tough and wouldn't give the best results. However, as long as we understand some code and have some experience with it, it can help us avoid further complex coding situations, either by coming up with creative solutions and dodging some features that are done through code or using some code and design decisions to make it happen. Adding a teleportation mechanic through code could be a bit complicated for those who can't code, but if we understand some basic coding experience we can still try to come up with a creative way to get through difficult situations like making a player change position or teleport in the game. It can be done by switching duplicated scenes and change player spawn location to make it look like the player is teleporting. Changing the scenes is done through a very simple line of code, and choosing the player spawn position can be done easily by dragging the player in the scene and placing him anywhere and the engine will automatically save his location. Or for example, if we want to make a maze game that is played by two players, both players represent half a heart and the goal is to make both players meet and connect in a way to complete the full heart under a time limit. It sounds complicated but it can be easily done and without any complex code, if both players had edge colliders that can be triggered by touching the other player, then all needed is to change the scene to the next, once they both connect in the right way.

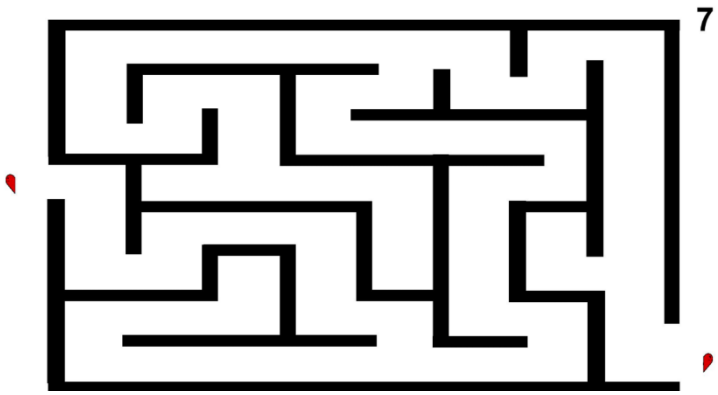

Figure 7 Example of no coded gameplay (Self-design game)

The picture above shows 2 players split in half in a shape of a heart, the goal is not to get out of the maze but to meet and connect inside of the maze, for this scene all needed is colliders on every object including players and 2 small trigger colliders on players that helps detect once players are connected and form the shape of a full heart. Instead of writing a load of code lines, we simply just add edge colliders that can be edited to fit a specific part of a player body and change scenes to the next with a small line of code once both players connect. There is always a chance that we end up getting stuck because we either forgot a line of code or just don't know how to code at all. In this situation we can try to look for answers online, there are unlimited sources where we can find solutions to our problems or get help from, such as YouTube, Reddit, Unity Scripting API, and much more websites. Making games without coding at all could be nearly impossible if that engine we were using did not provide features like visual scripting or other preprogrammed logic that allows developers to use them without the need for programming. Therefore it is recommended to get basic knowledge in programming to understand most of the basics which can allow you to make simple games and then you can try manipulating game engine tools or features besides coding few lines to finish implementing your ideas.

2.8 Visual scripting

Visual scripting is another perfect way to avoid coding a game, though it might require some time to get familiar with it, it helps most game developers to overcome and avoid complex coding.

"Visual Scripting is usually considered the act of creating a program using a summation of different small programs interacting with each other. There are different tools for Visual Scripting, which focus on reducing the difficulty of creating a program. With these tools, the

crafting of a program happens through an interface, and as a result of this, the user doesn't need to code but instead has to click, drag, and drop. This is the reason it is considered to be visual interaction with the tool, hence the term Visual Scripting. These tools, and the concept of Visual Scripting itself, targets animators, game designers, level designers, and artists who are looking to test their creation the way their users will see it. Of course, the scalability of this is not as great as coding the script ourselves. The more complex the features we want to create, the less chance there is to do it without writing any actual code. But if we want to stick to this to create not-so-complex games, this is our chance to do so quickly and easily" (Bertolini,2018). However some very complex games managed to use both visual scripting and some coding, they ended up having perfect results, one of those games is a 2D platformer known as "Hollow Knight" it has too many complicated designs and mechanics which made everyone think that the whole game was coded but in fact, it was a mix between programming and visual scripting.

2.9 Visual scripting Game engines

Many game engines aim to make games without a single line of code, Scratch is one of those engines that were and still popular for making simple games with only visual scripting. Anyone can use scratch even though it might be a little confusing to get familiar with it at the beginning, but in general, it is an easy engine to work with and understand and anyone with zero knowledge can make a game in it and upload it to their scratch website where players from around the world can play and learn from games that were made by other creators. There are many of these engines that are available for free nowadays but some of them can be complicated to use, and that is because every engine has a max capacity of how complex can visual scripting be in it, and for those engines that provide a lot of visual scripting, it could be hard to understand right away how to use them or what their functions are. Fortunately, most visual scripting engines are free to use and provide a good amount of tutorials on how to start using them.

2.10 Complex decisions

In some cases, we end up getting stuck while trying to implement a new feature or mechanic and neither coding nor any other feature or tool can help us in the engine to overcome the issue. So we start taking more time to decide whether we want to keep working on this project or just stop, or maybe change the project idea and stop implementing that complex mechanic or feature. Sometimes we try to work on an easier project that does not require a lot of experience in code. This all can lead to much different change results in a game. In some other cases, developers end up having another type of a complex decision, especially when the game story starts changing and some of the main characters die. “For example, changing story details and core design ideas too often for little reason creates an expectation of further changes. Developers don’t think they can count on anything staying the same, so they start avoiding investing themselves too much into ideas. They’ve had to suffer the emotional pain of watching their well-loved work die due to story changes too many times. The team’s personal investment and creative vibrancy slowly degrades” (Sylvester, 2013). In the end, every decision developers decide to take, will have it is own consequences and will impact that project differently. That’s why coming up with an idea can be much easier than implementing it, first, it has to fit the game project and second, it should not change the balance of the game. Finally what is most important is if the developer is capable of implementing it or has a way to make it work. Otherwise, it would be a bad decision to make a 3D shooting game if someone does not know how to code a shooting mechanic or make 3D models that fit with the project.

3. Conclusion

The research analyzed in which ways it is possible to use game design with less and without programming. This paper was aimed at exploring the different methods which are a valuable and creative alternative to coding in the game industry. For a better understanding of the different mechanisms, a theoretical basis was undertaken, where the key principles of a game engine and level design were discussed. As it was understood, game engines are complex tools that require detailed analysis.

Moreover, the paper explored a few of the various alternatives that can be used to avoid some level of coding procedures. In such a way, one of the central topics discussed were dedicated to manipulation in Game Engine Tools, Game Mechanics, and Level Design that in turn allow concluding how wide is the choice of ways that could be introduced in creating a game, which nowadays tends to overcome the traditionally used programming techniques. As stated from the research questions and the conducted analysis, there is a possibility to create a game without coding, however, it is essential to introduce visual scripting tools to succeed in this task. Besides, it is important to take into consideration that there is no concrete design theory to follow in creating a game, which gives developers the freedom of choice to make decisions that lead to different consequences.

The resulting paper allows questioning whether the future of coding is still relevant within the game industry or whether more and more ways to avoid it will appear. For instance, this theoretical research introduced in this paper will become the basis for the prospective final game project, which will prove the already discussed idea that it is now much easier to create games without complex theoretical and practical knowledge of programming.

Bibliography

Lewis, M., & Jacobson, J. *Game engines in scientific research - Introduction*. 2002

Andrade, A. *Game engines: a survey*. 2015.

Sylvester, Tynan. *Designing Games*. 2013.

Fabricatore, Carlo. *Gameplay and game mechanics design: a key to quality in videogames*. 2007.

Bertolini, Lucas. *Hands-On Game Development without Coding*. 2018.

Bleszinski, Cliff. *The Art and Science of Level Design*. 2000.