UNIVERSIDADE
BEIRA INTERIOR

Inside: A Video Game about Depression

Versão Final Após Defesa

Beatriz Brasil Costa Cabral Botelho

Dissertação para obtenção do Grau de Mestre em
Design e Desenvolvimento de Jogos Digitais

novembro 2020

Inside: A Video Game about Depression

Versão Final Após Defesa

Beatriz Brasil Costa Cabral Botelho

Dissertation submitted in candidature for the Degree of Master in
Game Design and Development
(2nd Cycle Degree)

Supervisor: Prof. Frutuoso Gomes Mendes da Silva

november 2020

Inside: A Video Game about Depression

Inside: A Video Game about Depression

Work developed at **Instituto de Telecomunicações**, 2020
under supervision of the **Prof. Frutuoso Silva**

How many times have people used a pen or paintbrush because they couldn’t pull the trigger? – Virginia Woolf

Acknowledgements

Working on this project was quite the adventure. In the beginning, I didn't feel quite sure I would be able to materialize the project I had in mind, because it seemed very ambitious and I'm not a proficient developer nor an artist. However, it is a pleasure to say that I did it with the help of my family, my friends, whom I consider my family as well, and my Professor and Supervisor, Professor Frutuoso Silva, for guiding me along this process. Therefore I want to thank you all, I did it because you believed in me and pushed me to conclude my dreams.

First, I want to thank my parents for being supportive and pushing me to do my best everyday. I would not have done it if it was not for their support and trust.

I want to thank my supervisor, Professor Dr. Frutuoso Silva, for accepting to be my supervisor and for helping me every time it was needed, giving me remarks and material to study and develop my game, which helped me improve and build a better video game.

I want to thank my best friend for supporting me and for always being there to help me with everything.

Lastly, I want to thank the rest of my family and friends for supporting me, not only during the development of this project but also during these last years.

This project is for all of you. I really hope you like it,

Beatriz Botelho

Resumo Alargado

Os videojogos fazem, cada vez mais, parte do quotidiano das pessoas. Mais importante, os jogos digitais têm-se construído, progressivamente, como produtos que desafiam o jogador moral e eticamente. Os videojogos não demonstram a sua influência apenas a nível moral e ético, influenciam também o comportamento das pessoas pelo próprio contexto social, económico e histórico em que se inserem. As escolhas que as pessoas realizam nos videojogos partem de uma contextualização moral, social, económica e histórica e importa compreender a relação que se desenvolve entre os jogadores, as suas contextualizações, e os videojogos, de forma a compreender de que forma os videojogos podem constituir-se como desafios morais e educacionais. Os objetivos desta dissertação são, portanto, destacar o potencial educacional que os jogos digitais evidenciam através da influência que podem ter no abalar, desafiar e consolidar dos sistemas morais das pessoas, e entender, adicionalmente, se os jogos digitais podem servir como ferramentas para a expressão daqueles que os desenvolvem. Isto é, procura-se descobrir o jogo digital enquanto objeto artístico que possibilita a expressão por parte do criador da obra. Esta dissertação tem como alvo criar um jogo sobre a depressão, de forma a construir não só um objeto de expressão educativo, como também um produto que possa ajudar doentes afetados pela depressão a lidar com a sua doença. Desta forma, o jogo procura informar as pessoas que o jogarem sobre a vida de uma pessoa que sofre de uma doença mental, neste caso, de depressão, elucidando-as sobre os riscos e sinais da depressão, permitindo que elas os reconheçam através desta aprendizagem. O jogo retrata a história de uma jovem que tenta cometer suicídio porque sofre de depressão e não consegue melhorar a sua condição. A jovem já tinha tentado melhorar a sua condição de várias formas, mas nunca conseguiu ver resultados. No fim, o suicídio pareceu-lhe a melhor opção. Depois de tentar acabar com a sua vida, a personagem acorda num mundo fantástico, desconhecido, que representa o coma em que se encontra submetida. Sirah, a personagem principal, terá de desvendar o mundo onde se encontra e tentar combater a sua depressão, numa batalha épica contra a doença. Para fazer isto, o jogador deverá controlar a personagem e desvendar a aventura sobre a sua vida, levando-a até à conclusão da sua história. Depois de conseguir derrotar a sua doença, Sirah acorda, percebe que estava adormecida num coma e compreende o momento de catarse que viveu. Assim sendo, o jogo desenvolvido pretende explorar o poder que os jogos digitais podem revelar na criação de momentos de reflexão, contemplação e desafio moral no jogador. No fundo, pretende-se averiguar se os jogos digitais têm o poder de educar através do desafio que apresentam perante os jogadores. O videojogo descrito nesta dissertação foi desenvolvido utilizando o motor de jogo Unity.

Palavras-chave

Depressão, Educação, Jogo Digital, Jogo Sério, Moral

Abstract

Video games are increasingly becoming a part of people’s daily lives. More importantly, digital games have been increasingly built as products that challenge the player morally and ethically. Video games do not only demonstrate their moral and ethical influence, they also influence people's behavior by their own social, economic and historical context. The choices people make in video games start from a moral, social, economic and historical contextualization and it is important to understand the relationship that is developed between players, their contextualization, and video games, in order to understand how digital games can constitute themselves as moral and educational challenges. Thus, it is intended to highlight the educational potential of digital games through the influence they can have on shaking, challenging and consolidating people's moral systems. The aim of this dissertation is to create a game about depression in order to build not only an educational object, but also a product of self-expression that can help depression-affected patients cope with their illness. The developed game aims to explore the power that digital games can reveal in creating moments of reflection, contemplation and moral challenges in the player. Basically, the aim is to find out if digital games have the power to educate through the challenges they present to players. The video game described in this dissertation was developed using the Unity game engine.

Keywords

Depression, Education, Mental Health, Morality, Serious Game, Video Game

Contents

List of Figures

List of Listings

Chapter 1

Introduction

1.1 Focus and Scope

The focus of this project is a video game that was built as a medium for self-expression and education about depression, as well as a way of coping for depression sufferers. The game was also developed in order to bring awareness to a disease that impacts more than 300 million people of all ages around the world (Global Health Metrics, 2018). Apart from this, it is also crucial that we understand if video games have the power to educate and be perceived as education tools.

It is imperative that we explore video games' power over the gamer (the one that chooses to engage with the digital game), understanding if and how video games can create reflection, contemplation and moral challenges for its players.

It is commonly known that video games are increasingly becoming a part of people's daily lives. If in 2003 Squire showed us that video games had rapidly become one of the most widespread, lucrative and influential forms of entertainment in the United States and around the world (Squire, 2003: 2), today, the scenario is quite wider. Today, it is through digital games that people socialize, unwind after a long day at work, find moments of catharsis and expression, and, most importantly, live alternative versions of themselves (recall The Sims game series (Electronic Arts, 2000: first title), a life simulator). According to Wolf,

> Video games have moved, possibly surpassing even movies, into a central role in (...) popular culture in a relatively short time, and today there is increasing evidence that the video game console – to some extent, as much as the personal computer – has emerged as a central media device (2017: 1)

Video games are already a prevalent subculture of the new media and are, above all, designated as an experience that requires the players to share much of themselves. The player is the driving force of action that develops within the virtual world; the player is the one who makes all the choices and actions that seek an end: the end of the game. For this reason, it should be noted that video games contain times when the player must act in accordance with their moral and ethical values. Therefore, it is important to analyze the influence these playable products have on players' moral and ethical plans. In addition, it is important to analyze the behaviors players have within the virtual worlds in order to understand the ways they use these same worlds. The authenticity of a player's involvement is proven when, as a form of example, we compare human involvement with shows and movies, which can accumulate very high audience levels. If while watching a show or a movie passively, oblivious to the action because the spectator is out of it, in a video game, the action is in the hands of the one who decides to play it. The result is starred by the ones who decidedly involved themselves emotionally, morally and ethically with the game. According to Zagal,

> Computers (…) allow their users to play equivalent roles to both the drama performer as well as the audience member. (…) What this means is that since games provide play spaces where people not only transform the gameworld, but also themselves, they can be used to explore ethical reasoning. When used as a transformative tool, video games can empower people to learn what it means to live ethically and how to go about doing so (2009: 1)

That said, we present a set of investigation questions that will guide the architecture and purpose of this dissertation: What can video games say about people's moral and ethical compositions? Can video games be strong allies of education through their representative and challenging power? Moreover, are games strong forms of expression for those who develop games? Basically, it is intended to understand what kind of influence video games can have on people's moral and ethical plan, and how this influence can be used as a form of education. Moreover, we also want to understand if video games are, indeed, forms of expression.
Obviously, if there is influence, it is necessary to develop documentation that technically, scientifically and emotionally informs players not only in a pre-game context, but also in a game and post-game context; and developers in their development process.

1.2 Problem Definition and Objectives

The main objectives for this dissertation were:

- To investigate if video games can become powerful educational tools
- To investigate if video games can represent powerful expression tools
- To investigate if video games can be powerful coping mechanisms used by depression sufferers
- To develop a prototype of the game using Unity

In order to accomplish these objectives, there was the need to perform some tasks, such as the elaboration of a state of art about games that challenge players, especially the ones about depression developed with educational purposes. The state of art, or the related work (Chapter 3), was the first step to be taken. Many video games about depression were found but we had to choose the ones that were more relevant to this dissertation. After this, the video game was developed. Unity was the game engine chosen to develop the digital game.

1.3 Organization of the Dissertation

This dissertation is divided into four main chapters preceded and succeeded by the Introduction and Conclusions and Future Work, respectively. In the sixth chapter, the bibliographical references used on this dissertation can be found, followed by a couple of attachments. The contents of each chapter of this dissertation can be summarized as follows:

Chapter 1 is an introduction of the overall project, explaining the context of the dissertation. It also includes the delimitation of the main objectives and the problem to be solved.

Chapter 2 offers an overview about serious video games and their potentials. In this section, we can also understand how video games can challenge gamers and how this challenge can represent an educational purpose. This said, we will additionally try to offer an overview on how video games can be powerful expression tools.

Chapter 3 Video Games about Depression and their purposes are presented in this chapter, followed by some of the related work.

Chapter 4 This chapter is the main core of the dissertation as it presents the video game and the process through which it was developed. Furthermore, we explain the reasons why some decisions were made and the implementation of the game's mechanics. Chapter 5 The final chapter wraps up the conclusions of this dissertation and the directions for future work.

Chapter 2

Serious Video Games

2.1 Learning Through Challenges

Video games have evolved through time. They went from a form of entertainment appreciated by a few enthusiasts to richly immersive worlds enjoyed by large amounts of people around the globe. According to Eichenbaum et al. (2014: 50), video games "have become big business, bringing in more than sixty billion dollars in global revenue annually". Not only have they become big strings of money, they have also been "increasingly recognized as an artistic medium (...), as institutions such as the Museum of Modern Art and the Smithsonian (...) feature them" (Eichenbaum et al., 2014: 50). Furthermore, "games have evolved as novel and effective teaching tools" (Eichenbaum et al., 2014: 50). Today, video games represent more than mere entertainment objects. "They also fight against declining mental capacities in old age, promote job-related skills, and offer models of how to teach children complex tasks and abilities" (Eichenbaum et al., 2014: 50). Video games prosper not only because of good marketing and advertising nor the ready availability in the market. Video games prosper because they are enjoyable (Eichenbaum et al., 2014: 50). More interestingly, "The fun in playing video games depends largely on the game's effectiveness in teaching players to succeed" (Eichenbaum et al., 2014: 50). Video games also need to "teach players to succeed on a set of tasks that are initially quite difficult" (Eichenbaum et al., 2014: 50). Consumers would not want to play video games if they had already mastered them from the start; nor would they want to play video games they could never master (Eichenbaum et al., 2014: 50). With this in mind, we can see clearly that game designers "need to ensure that players do fail (if they always succeed, there is nothing to learn) but(...) this failure does not seem insurmountable (...)" (Eichenbaum et al., 2014: 50).

According to Eichenbaum et al.,

> (...) modern video games instantiate and demonstrate many key principles that psychologists, educators, and neuroscientists believe enhance learning and brain plasticity. This, in turn, has led some scientists to consider the possibility that video game play may produce not just improvements in the ability to play the games but may, in fact, result in more fundamental changes in the way players see the world and process information. It has also led to a burgeoning field that uses video games—even those designed purely for entertainment—in a variety of practical applications, from training individuals with cognitively and perceptually demanding jobs (e.g., military pilots and endoscopic surgeons) to rehabilitating individuals with deficits in particular types of visual or

> cognitive processing (e.g., individuals with amblyopia or with dyslexia). In all, what began as simple play has become something with significant real-world relevance (2014: 51)

As we can see, video games have become more than mere entertainment tools, having accomplished a new form of experience: education through challenge. Video games as a form of education appeal to people because of the video game feature. According to Malone apud Blumberg, "This appeal, which is found among a diverse age range and among both females and males, has been attributed to the curiosity, fantasy, and challenge inherent in game play (...)" (2014: 4). However, and more recently, other features have appeared and contributed to the appeal of video game play. We cannot forget that there are a lot of features that contribute to the appeal of video games that will not be investigated here. However, we will cover the features that seem to sustain the purpose of this dissertation. As such, we can find "(...) interactivity, whereby players initiate and receive feedback about their actions, which affects their game play experience" (Renkl & Atkinson, 2007; Ritterfeld et al., 2009 apud Blumberg, 2014: 4); "agency or control, which refers to players' ability to manage aspects of their game play, such as the use of the control mechanisms or the unfolding of the story line" (Skalski, Lange, Tamborini, Helton, Buncher & Lindmark, 2011; Qin, Rau & Salvendy, 2009; Wood Griffits, Chappell & Davies, 2004 apud Blumberg, 2014: 4); "identity, which refers to players' opportunity to form relationships and linkages with game characters or to become game characters via avatar construction" (Blascovich & Bailenson, 2011; Lane et al., 2013; Trepte, Reinecke, & Behr, 2010 apud Blumberg, 2014: 4); "feedback, which refers to the information players receive about the efficacy of their game actions, which in turn scaffolds the course of their game play" (Lane et al., 2013; Lieberman, 2006 apud Blumberg, 2014: 5); and " immersion, which refers to players' sense of presence or integration within the game" (Tamborini & Skalski, 2006 apud Blumberg, 2014: 5). These features promote enjoyable game experiences, and in the context of educational game play, increasing game play time often reveals more opportunities for learning. "As every educator knows full well, one of the best predictors of learning is time on task" (Eichenbaum et al., 2014: 51). It is obvious that "more hours spent on a task means more learning, and, clearly, video games excel at encouraging players to put time on task" (Eichenbaum et al., 2014: 51). Not only that, but video games are "capable of providing a variety of basic psychological needs" (Eichenbaum et al., 2014: 52), as stated by behavioral research. Among these, and according to Przybylski, Rigby, and Ryan 2010 apud Eichenbaum et al., we can find

> autonomy (the belief that one has control over his or her own actions and decisions), competence (the the belief that one has the level of skill necessary to achieve goals), and relatedness (the feeling that one is socially connected with other human beings) (2014: 52)

As we can see, interactivity, agency, identity, feedback and immersion are key features when building a game as an educational experience. This is important because we can understand that when players are able to have control over what occurs inside and during game play, understand their actions alter the game experience, feel linked to the game characters, receive information about the efficacy and impact of their game actions and feel part of the game, emerged in its' world, players become available to learn, reconsider and reiterate their inside worlds, building new ones. This is education: experiencing something new and becoming someone new due to the life changing experiences.

2.2 Video Games as a form of Expression

In this section, we will reveal how video games can work as a way through which game designers and developers can express themselves.

2.2.1 Video Games as an art form

As stated above, and according to Eichenbaum et al., video games have been increasingly recognized as an artistic medium. However, what is art, what does it represent, and does it mean expression?
It is quite difficult to define art and we're not looking towards establishing a central and perfect definition of art. However, we can try to examine what art professionals think of it, in order to understand it better ourselves. According to Freeland, an american art philosopher, "Many modern artworks challenge us to figure out why, on any theory, they would count as art" (2001: 18). We can understand that art is a quite difficult subject to define with a single interpretation or representation. On this matter, Freeland shows us what art can be: "art is an imitation of nature or of human life and action" (2001: 31), art could also be "any artifact ... which has had conferred upon it the status of candidate for appreciation by some person or persons acting on behalf of a certain social institution" (2001: 55); art could also be "an object that embodies a meaning" (2001: 57); "art is the best possible window into another culture" (2001: 63); "art is a very deep expression of attitudes and outlook" (2001: 88), but most importantly: "art is not exhausted by one theme" (2001: 146). We can understand that art is not represented by one single identity, and, in this reality, we can only offer our personal and individual look on it. In order to do this, we will dive deeper inside the subject we are covering, and investigate further

along game development and design and its' relation with art. Crawford, a video game designer, states that

> (...) art is something designed to evoke emotion through fantasy. The artist presents his audience with a set of sensory experiences that stimulates commonly shared fantasies, and so generates emotions. Art is made possible only by the richness of the fantasy world we share (2000: 2)

However, there are many practical problems associated with stimulating fantasies inside another person's mind. "A major problem is getting the attention or participation of the audience. Most art allows very little participation" (Crawford, 2000: 3). If we think about how people enjoy a piece of music, a performance or even paintings and other types of art that are available in a museum or galeria, we realize that people enjoy these art forms passively. According to Crawford,

> The artist does all the active work, makes the biggest emotional investment. The audience is expected to absorb quietly the fruits of the artist's exertions. Active participation is severely curtailed. Without participation, attention dwindles and impact crumbles away (2000: 3)

Viewed through these lenses, art, or the traditional way through which art is expressed, is not very participatory, leaving no space for the audience to engage with the art object actively. Leaving all the work in the hands of the creator and expecting the audience's passivity could possibly result in a less powerful experience of the ones interacting with the art piece. How do video games change this scenario and welcome people to be a part of the artwork? During the introduction age of computers, computers didn't do more than crunching numbers. However, when graphics and sound capabilities were joined to the other competences, computers assumed a new personality. According to Crawford,

> These capabilities gave the computer a powerful asset: it could now communicate with the human, not just in the cold and distant language of digits, but in the emotionally immediate and compelling possibility: the possibility of using the computer as medium for emotional communication art. The computer game has emerged as the prime vehicle for this medium. The computer game is an art form because it presents its audience with fantasy experiences that stimulate emotion (2000: 3)

If we think about video games as an art form, then we can think that the ones who design and develop the games are potentially artists. It is a possibility. Nonetheless, video games became a different art form as they invited people to be a part of the art itself. With access to a computer, the artist has a tool that is more subtly indirect than traditional art. With other art forms, the artist directly creates the experience that the audience will encounter. This last art form relies on the non participation of people, as they are outside of the art. "Since this experience is carefully planned and executed, the audience must somehow be prevented from disturbing it; hence, non participation" (Crawford, 2000: 3). Nonetheless, "Video games allowed artists to tackle a more difficult sub-problem facing non-performed arts, the problem of how to involve the audience in mechanically reproduced art" (Smuts, 2005:7). In a video game, "the artist creates not the experience itself but the conditions and rules under which the audience will create its own individualized experience" (Crawford, 2000: 3). In this case, the demand on the artist is higher, once the experience is planned indirectly, taking into account the probable and possible actions and reactions of the audience. The return is far greater since participation increases attention and heightens the intensity of the experience (Crawford, 2000: 3). According to Crawford,

> When we passively observe someone else's artistic presentation, we derive some emotional benefit, but when we actively participate in a game, we invest a portion of our own ego into the fantasy world of the game. This more sizable investment of participation yields a commensurately greater return of emotional satisfaction (2000: 3)

Therefore, video games, being intrinsically participatory, present the artist with a fantastic opportunity for reaching people (Crawford, 2000: 3).
Opposed to Crawford, who believes game developers and designers create the rules and conditions that allow players to build their own experiences, Smuts believes that video games are more than rules:

> Though video games appear to be performative, what might count as the performance-the playing-is not considered art. Perhaps this is because the games themselves draw more attention than the players. (...) Electronic games are different in that they are much more than rules. They include narratives, graphic design, characterization, dialogue and more (2005: 6)

Video games became capable of allowing shared experiences and interactivity between the art product and the audience, bringing importance to both game creator and game player. According to Smuts,

> We often hear it said that films can "break the fourth wall" through techniques such as directly addressing the audience, but the wall remains. It is ontologically impossible for the audience of a film to break the wall. Video game technology has allowed artists to experiment with solutions to the problem of how to make an interactive movie: Video games are the first concreative mass art. (2005: 8)

Video games allow game designers and developers to express themselves in a product that is shared with the people who play it. It takes two to enjoy the game as an art form: the creator and the player. We can think of video games as art because they can be a form of expression, and, for some, expression is art, and art can represent expression. However, video games are different in a matter of participation and interactivity: the game is a shared experience between the ones who created it and the ones who play it and experience it. Both creator and player can express themselves in a video game: the creator developing a game as an expression and the player playing a game, interacting and connecting to a game as an expression of themselves. The players can express themselves in the video game by playing it the way they desire, and every reaction, emotion or any other event that could happen due to this interaction could indeed be a moment of expression.

Furthermore, and according to Smuts,

> Theorists who call themselves ludologists, argue that video games should not be considered just another narrative art form, but a form of play. Other theorists, narratologists such as Janet Murray, argue that video games can and should become more narrative-driven in order to realize their artistic potential. This seems to be the path game developers have chosen. Current video games have highly integrated narratives that are often far more complex than the most sophisticated noir plots (2005: 8)

We can see that much like film grew out of photography and drama, video games grew out of digital animation. Games share narrative themes and expressive goals, just like movies do (Smuts, 2005: 11). The interactive capability associated with the narrative feature that has been established in video games has brought out the artistic potential of video games, through the arousal of emotions in the audience (Smuts, 2005: 12). Remember video games such as That Dragon Cancer (2016, Numinous Games), a game about a father dealing with his twelve months old son's cancer. Even though many games are more clearly about triumphant victory in battle, nothing stops game designers and developers from creating games about the less good parts of life. Current narrative-based video games can put players in the position of the one who makes the decisions that affect the lives of the people inside the narrative. In doing so, video games become strong ways to comment, express and resemble an object of catharsis for the game creators via fictional representation. This way, both the creators and players can feel the game and express themselves through it.

Chapter 3

Related Work

3.1 Video games about Depression

Video games about depression have been developed over the years with the intention of educating people about the disease and its symptoms, not only helping depression sufferers deal and cope with their illness, but also representing a piece of cathartic work done by the game developers themselves. We have been seeing the release of a new wave of video games that talk about mental health themes and portray them in everyday life situations, such as Life is Strange (Dontnod Entertainment, 2015). Many of these games are helpful in bountiful ways. They can help enlighten people about the symptoms of the disease, so that they can be better informed and reach out when they encounter themselves against the red flag signs.

Video games like these can represent a way through which people can relieve their troubles and feel less lonely. Not only this, digital games can also be a tool used by game developers to express and translate their experiences with the disease. This said, video games about depression can not only serve as an educational tool, but a tool that allows healing through expression as well.

3.1.1 Depression Quest

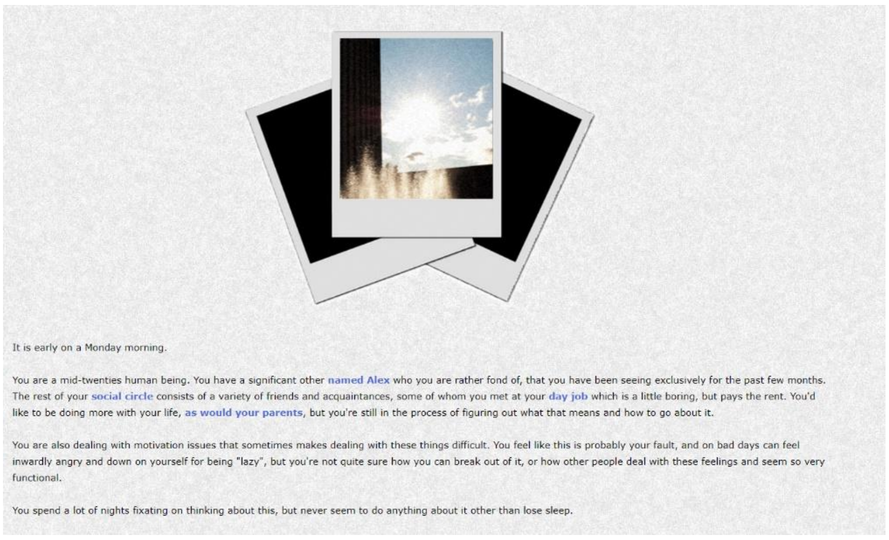
Depression Quest is a text-based game developed by Zoe Quinn, Patrick Lindsey and Isaac Shankler, available to play on the web since 2013 (Quinn *et al.,* 2013). The game explores the nature of depression, allowing people who have never suffered from the illness to experience this condition that affects so many people worldwide. In the game, the player can understand how depression can harm one's mood, energy and motivation. 40 thousand words are all that it takes for the player to know and understand what it is like to go through a day as someone who suffers from depression. The game tries to put the player in the shoes of another, a person who is ill, creating a request for empathy and learning through it. According to the authors of the game,

> Depression Quest is a game that deals with living with depression in a very literal way. This game is not meant to be a fun or lighthearted experience. (...) The goal of this game is twofold: firstly, we want to illustrate as clearly as possible what depression is like, so that it may be better understood by people without depression. Hopefully this can be something to spread awareness and fight against the social stigma and misunderstandings that depression sufferers face. Secondly, our hope

> is that in presenting as real a simulation of depression as possible, other sufferers will come to know that they aren't alone, and hopefully derive some measure of comfort from that (Depression Quest, 2013).

In this game, the player can be part of a simulation and go through multiple stages of a day and the life of a person who suffers from depression. The game starts by introducing you to the simulation (Fig.1). It sets a ground for you to live the game in first person, and as the person they describe:

> You are a mid-twenties human being. You have a significant other named Alex who you are rather fond of, that you have been seeing exclusively for the past few months. The rest of your social circle consists of a variety of friends and acquaintances, some of whom you met at your day job which is a little boring, but pays the rent. You'd like to be doing more with your life, as would your parents, but you're still in the process of figuring out what that means and how to go about it. You are also dealing with motivation issues that sometimes makes dealing with these things difficult. You feel like this is probably your fault, and on bad days can feel inwardly angry and down on yourself for being "lazy", but you're not quite sure how you can break out of it, or how other people deal with these feelings and seem so very functional. You spend a lot of nights fixating on thinking about this, but never seem to do anything about it other than lose sleep (Depression Quest, 2013)


It is early on a Monday morning.

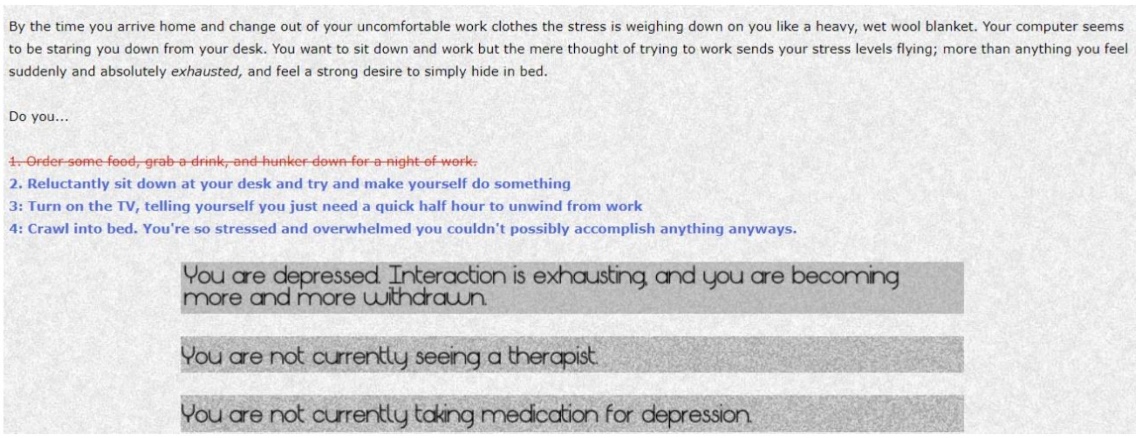
You are a mid-twenties human being. You have a significant other **named Alex** who you are rather fond of, that you have been seeing exclusively for the past few months. The rest of your **social circle** consists of a variety of friends and acquaintances, some of whom you met at your **day job** which is a little boring, but pays the rent. You'd like to be doing more with your life, **as would your parents**, but you're still in the process of figuring out what that means and how to go about it.

You are also dealing with motivation issues that sometimes makes dealing with these things difficult. You feel like this is probably your fault, and on bad days can feel inwardly angry and down on yourself for being "lazy", but you're not quite sure how you can break out of it, or how other people deal with these feelings and seem so very functional.

You spend a lot of nights fixating on thinking about this, but never seem to do anything about it other than lose sleep.

Fig.1: The player's first encounter with the game. Source: Depression Quest (2013)

After this, the player is requested to click a piece of text that says "Next" so that they can go through the simulation. Things don't seem to be that well with the character, they feel down, unmotivated and stressed. The player can go through all this text and start to enter the skin of the ill person. This merge between character and player is accentuated when the player is requested to make a choice. Out of four seemingly available choices, one is unavailable. The player is meant to choose one out of three of these decisions (Fig.2): "2. Reluctantly sit down at your desk and try and make yourself do something, 3: Turn on the TV, telling yourself you just need a quick half hour to unwind from work, 4: Crawl into bed. You're so stressed and overwhelmed you couldn't possibly accomplish anything anyways".


By the time you arrive home and change out of your uncomfortable work clothes the stress is weighing down on you like a heavy, wet wool blanket. Your computer seems to be staring you down from your desk. You want to sit down and work but the mere thought of trying to work sends your stress levels flying; more than anything you feel suddenly and absolutely *exhausted*, and feel a strong desire to simply hide in bed.

Do you...

~~1. Order some food, grab a drink, and hunker down for a night of work.~~
2. Reluctantly sit down at your desk and try and make yourself do something
3: Turn on the TV, telling yourself you just need a quick half hour to unwind from work
4: Crawl into bed. You're so stressed and overwhelmed you couldn't possibly accomplish anything anyways.

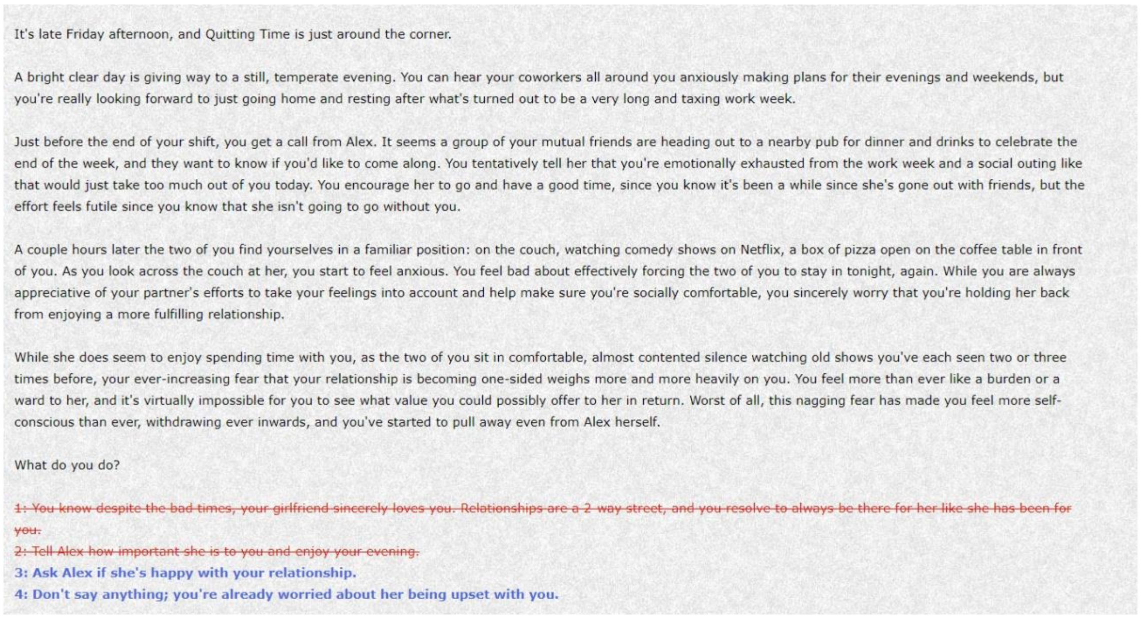
You are depressed. Interaction is exhausting, and you are becoming more and more withdrawn.

You are not currently seeing a therapist.

You are not currently taking medication for depression.

Fig. 2: The players' encounter with the first choice to be made in the game. Source: Depression Quest (2013)

The game forfeits choices from the player, as a way of making the player feel as if one's trapped (a symptom of depression). Out of an array of options, the player is never allowed to choose between all of them. This game mechanic reminds that depression is not a state of mind, neither can it be postponed nor turned off. It is an illness that is constant, that prevents people from living a normal and happy life. In this term, the game does a great job portraying the impediments that live inside a depressed person's mind and body. The "normal" or "healthy" choice is always unavailable, which represents depression through limitation. This narrowness makes the player understand the mind of a depressed person. The burdens of carrying interpersonal relationships while being depressed are also depicted in this video game (Fig.3).


It's late Friday afternoon, and Quitting Time is just around the corner.

A bright clear day is giving way to a still, temperate evening. You can hear your coworkers all around you anxiously making plans for their evenings and weekends, but you're really looking forward to just going home and resting after what's turned out to be a very long and taxing work week.

Just before the end of your shift, you get a call from Alex. It seems a group of your mutual friends are heading out to a nearby pub for dinner and drinks to celebrate the end of the week, and they want to know if you'd like to come along. You tentatively tell her that you're emotionally exhausted from the work week and a social outing like that would just take too much out of you today. You encourage her to go and have a good time, since you know it's been a while since she's gone out with friends, but the effort feels futile since you know that she isn't going to go without you.

A couple hours later the two of you find yourselves in a familiar position: on the couch, watching comedy shows on Netflix, a box of pizza open on the coffee table in front of you. As you look across the couch at her, you start to feel anxious. You feel bad about effectively forcing the two of you to stay in tonight, again. While you are always appreciative of your partner's efforts to take your feelings into account and help make sure you're socially comfortable, you sincerely worry that you're holding her back from enjoying a more fulfilling relationship.

While she does seem to enjoy spending time with you, as the two of you sit in comfortable, almost contented silence watching old shows you've each seen two or three times before, your ever-increasing fear that your relationship is becoming one-sided weighs more and more heavily on you. You feel more than ever like a burden or a ward to her, and it's virtually impossible for you to see what value you could possibly offer to her in return. Worst of all, this nagging fear has made you feel more self-conscious than ever, withdrawing ever inwards, and you've started to pull away even from Alex herself.

What do you do?

~~1: You know despite the bad times, your girlfriend sincerely loves you. Relationships are a 2-way street, and you resolve to always be there for her like she has been for you.~~
~~2: Tell Alex how important she is to you and enjoy your evening.~~
3: Ask Alex if she's happy with your relationship.
4: Don't say anything; you're already worried about her being upset with you.

Fig. 3: The player must make a decision regarding another issue that came by. This time, the character is worried about his relationship. Source: Depression Quest (2013)

As the player goes forward in the simulation, more and more problems seem to accumulate. The character that the player impersonates goes through a long road of challenges that they find themselves incapable of dealing with. Overall, as a simulation of someone who is depressed, the game does a good job making the player go through the burdens of a depressed person, such as anxiety, depersonalization, feelings of not being good enough, and constant fatigue.

3.1.2 Actual Sunlight

Actual Sunlight is a cross-platform game developed by WZOGI, available to play on Steam since 2014, that depicts the story of Evan Winter, his depression and suicide ideations (WZOGI, 2014). The character is infected with loneliness, severe depression and suicide thoughts, and has to balance his work life with these issues. The game is almost completely presented in text form that represents the thoughts of the character. The player gets to know Evan through his thoughts and sometimes quirky remarks of everyday life. The availability of his thoughts are also the reason why the player comes to realize that Evan is deeply sick. Everyday life tasks are a burden, everything feels like an impossible task and life isn't going well (Fig.4).

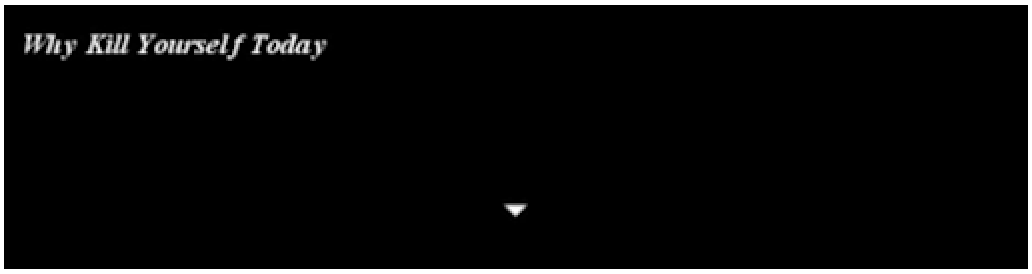

Why Kill Yourself Today

Fig. 4: Evan's Thoughts. Source: Actual Sunlight (2014)

Depression is well translated in many instances of the game. Depressed people come to a point in life where there is a constant craving to become healthier, a desire to start a new life and become capable of being normal. For example, Evan tries many times to leave his room but finds himself incapable of doing so (Fig.5).

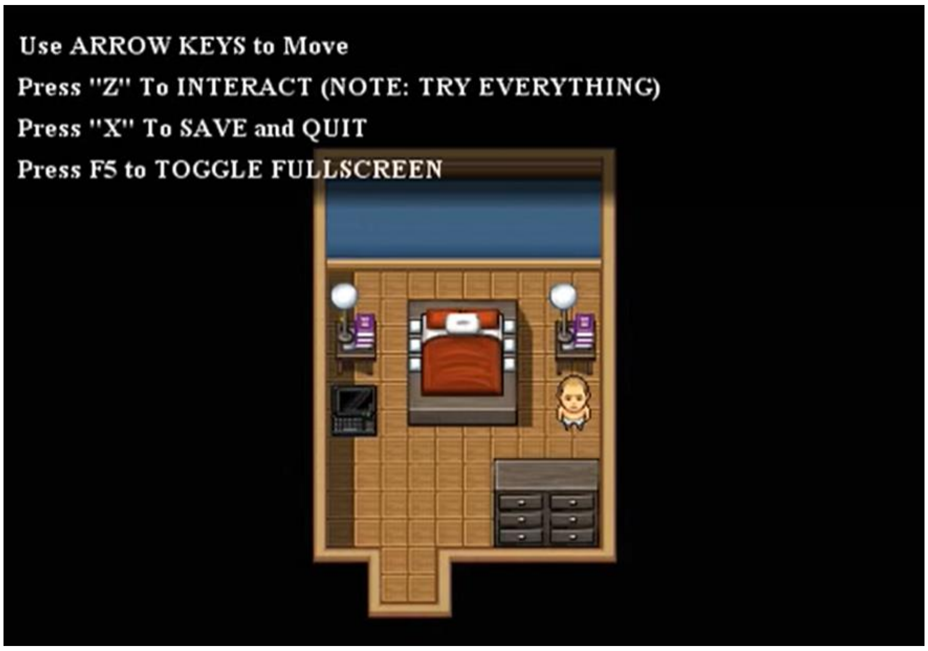

Use ARROW KEYS to Move
Press "Z" To INTERACT (NOTE: TRY EVERYTHING)
Press "X" To SAVE and QUIT
Press F5 to TOGGLE FULLSCREEN

Fig. 5: Evan tries to leave his room. Source: Actual Sunlight (2014)

In conclusion, this game tries to portray that dealing with depression means fighting to survive and surviving because of the deep hope of getting better one day. Actual Sunlight does a great job at representing all these issues, letting the player read through the mind of Evan and understanding his life, and, therefore, understanding how the mind of a mentally ill person works.

3.1.3 To Leave

To Leave is a game developed by Freaky Creations, available to play on Steam since 2018, that depicts the story of Harm, a young man suffering from manic depression who possesses a magical door (Creations, 2018) (Fig.6). One must not forget that depression comes in various types: Clinical Depression, Dysthymia, Postpartum Depression, Psychotic Depression, Atypical Depression, Situational Depression, Disruptive Mood Dysregulation and even Manic Depression. It is important that video games approach the different types of the disease so that they can be better portrayed and understood, and this game does this well.Video games like these can represent a way through which people can relieve their troubles and feel less lonely. Not only this, but digital games can also be a tool used by game developers to express and translate their experiences with the disease. This said, video games about depression can not only serve as an educational tool, but a tool that allows healing through expression as well.

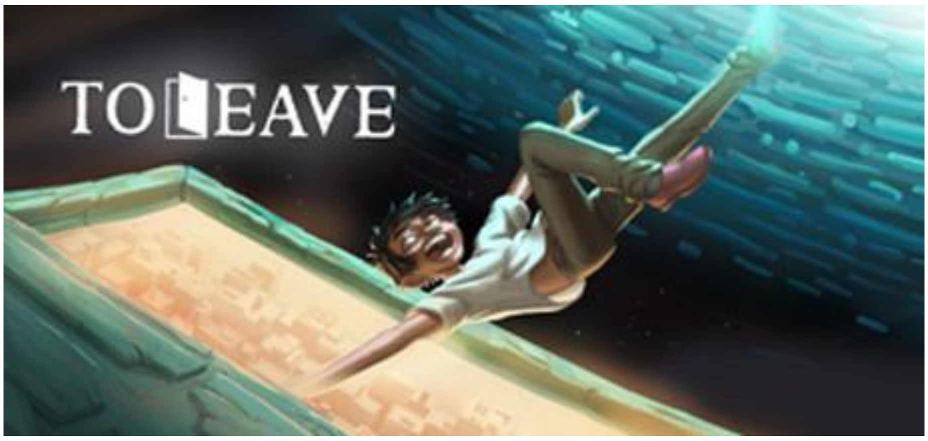


Fig. 6: To Leave game art. Source: To Leave (2018)

3.1.4 Distracting Myself From Suicide

Distracting Myself From Suicide is a semi-autobiographical video game about crippling depression and the coping mechanisms the authors of the game found to distract themselves from it (Robot, 2017). The game was developed by Giant Evil Robot and released in 2017. The player controls a man who has a camera that is being used to take Instagram pictures. As the player goes through the snowy forest, looking for good spots to take a picture, dark thoughts seem to manifest in the mind of the character (Fig.7).

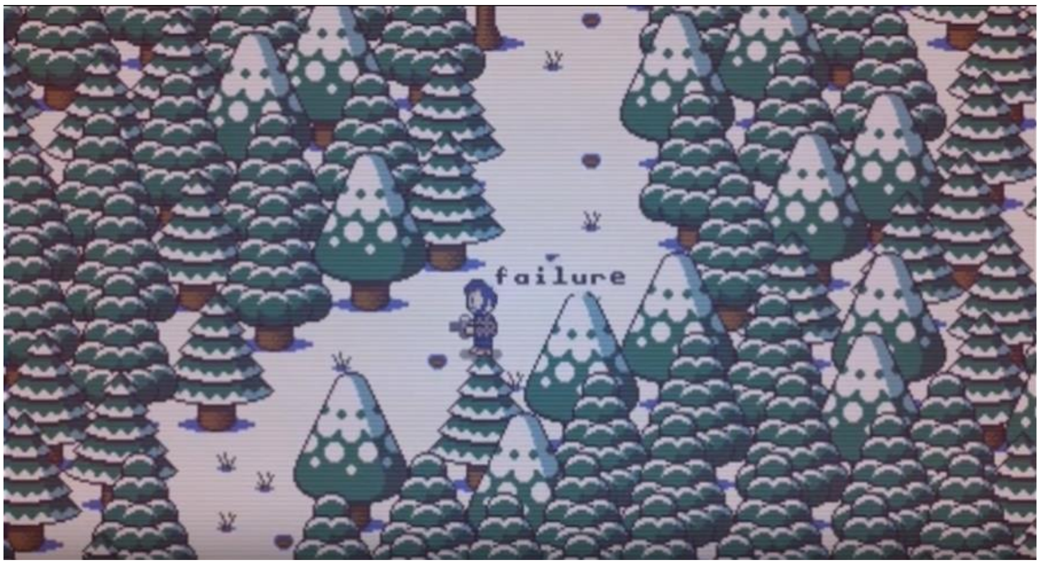


Fig. 7: Dark thoughts manifesting in the mind of the character. Source: Distracting Myself From Suicide (2017)

Sometimes the dark thoughts come out as “failure”, other times “idiot”, “loser”, “alone”, “awkward”, “unlovable”, “unwanted” “hopeless”,even “worthless”. As the player progresses in the game, the dark thoughts seem to appear more often and in a shorter amount of time. Just before the end of the game, the dark thoughts manifest abruptly and fast, leaving no space for the player to continue on the trail taking pictures. The thoughts eventually come too much to bear and the screen goes black.
In conclusion, this game represents how depression and negative thoughts infest people when they are just trying to live a normal life.

3.1.5 What Now?

What Now? is a simulation based video game about depression, anxiety, hopelessness and PTSD (Grimes, 2014). It was developed by Arielle Grimes, also known as slimekat, and released in 2014. When the character gets close to certain objects, such as the personal computer or the bed, they share their thoughts (Fig.8).

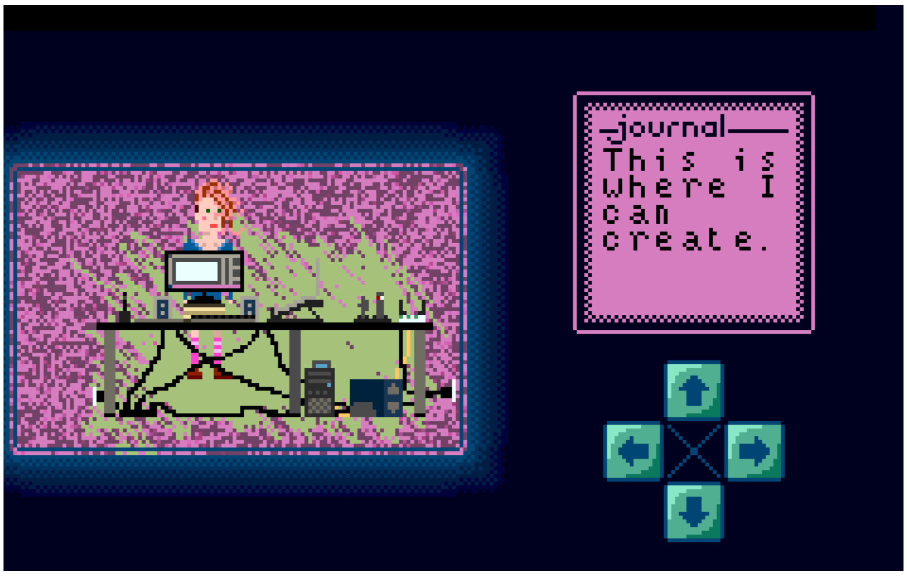


Fig. 8: The character shares their thoughts. Source: What Now? (2014)

As the player approaches the end of the game, glitches start to appear, the music becomes noisy and the thoughts become loud, infesting the whole screen (Fig.9). The game finishes with a question: “What Now?”.

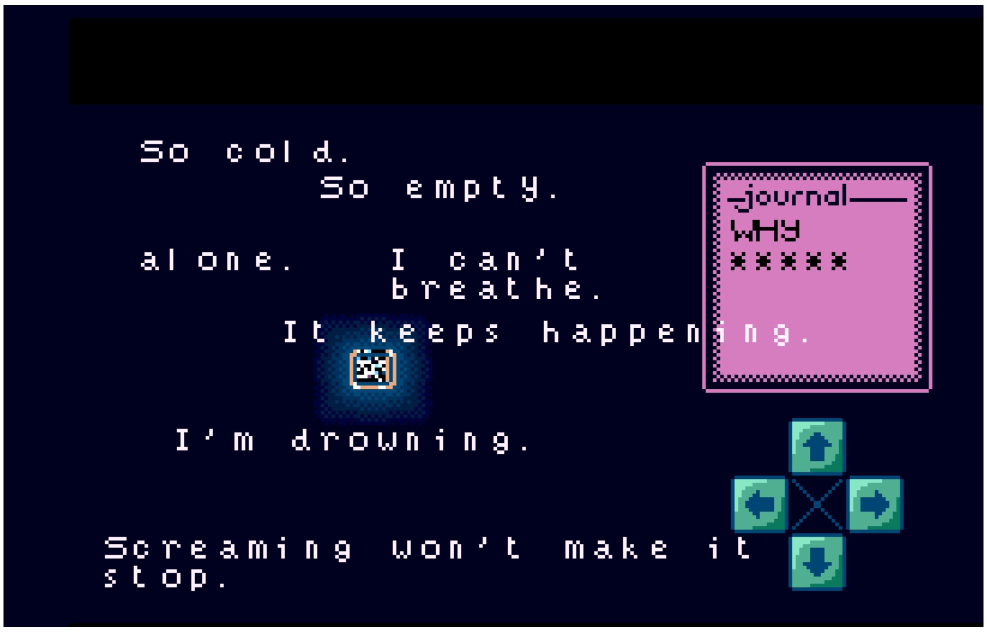
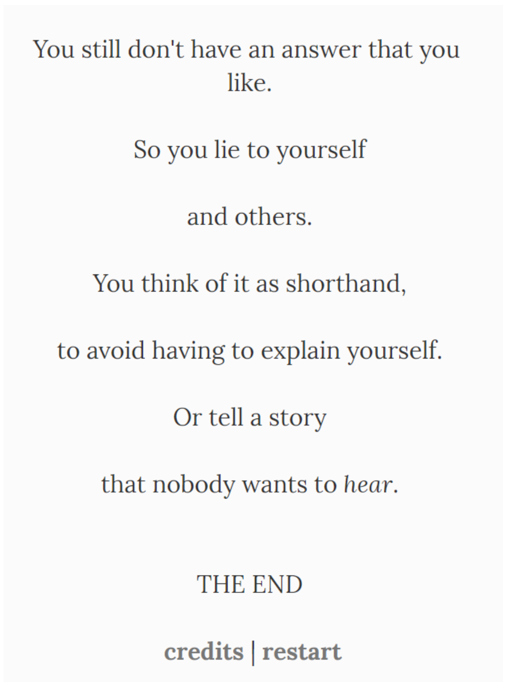


Fig. 9: The character’s dark thoughts spread around all of the game screen and then disappears between the black matter and glitches. Source: What Now? (2014)

What Now? serves, predominantly, as a representation of a moment that many people who suffer from any mental illness can probably identify with - not knowing what to do when one realizes one is sick and not being able to act about it. It informs the player about one stage of the life of a mentally ill person, and for this reason, educates people.

3.1.6 I’m better

I’m better is a text based video game about depression, suicide and trauma, develped by Jeremy Lonien, available to play on itch.io (2014). The game portrays the thoughts of a depressed person who is in a party talking with some friends and family members (Lonien, 2014). The only interaction possible with this game happens when the player clicks the text in order for it to progress. The text represents the character’s thoughts, and the player, who has access to their thoughts, discovers that the character has been lying about their mental health for years (Fig.10).


You still don't have an answer that you like.

So you lie to yourself

and others.

You think of it as shorthand,

to avoid having to explain yourself.

Or tell a story

that nobody wants to *hear*.

THE END

credits | **restart**

Fig. 10: The character lies about their mental health. Source: I’m better (2014)

The character beats themselves over the fact that they do not understand why they aren’t healthy, why they haven’t gotten better yet, when they have tried so many methods to achieve this. Since the character doesn’t understand themselves why they aren’t healthier, they also lie to their family and friends, dismissing the subject with the answer “I’m better” (Fig.11).

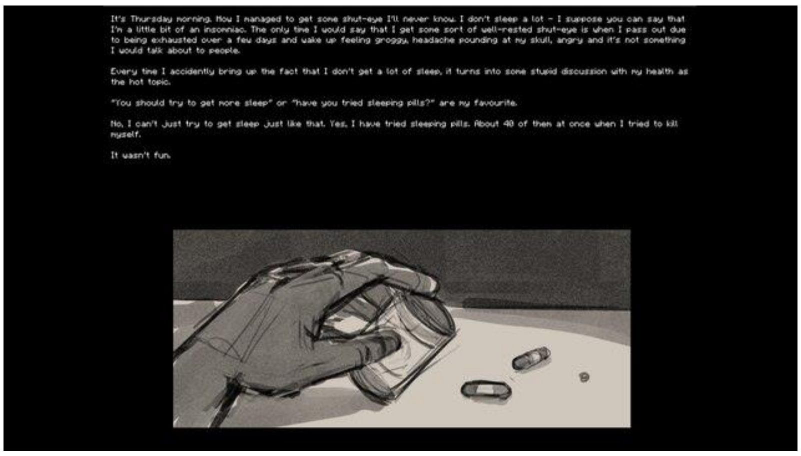
“I’m better.”

Fig. 11: “I’m better.”. Source: I’m better (2014)

This game retrieves the importance of having a strong emotional support group. Many times, people don’t understand why they are sick, even after trying many methods to get better. This can result in fear of sharing the truth with the social groups they find themselves in, and this game depicts this.

3.1.7 I’m Fine

I’m Fine is a video game that shares an interactive narrative experience about a man who suffers from depression and suicidal thoughts (Rokashi, 2016). The game goes through his life and how his one true desire is to be accepted. It was developed by Rokashi and was released in 2016 (Fig.12).


It's Thursday morning. How I managed to get some shut-eye I'll never know. I don't sleep a lot - I suppose you can say that I'm a little bit of an insomniac. The only time I would say that I get some sort of well-rested shut-eye is when I pass out due to being exhausted over a few days and wake up feeling groggy, headache pounding at my skull, angry and it's not something I would talk about to people.

Every time I accidently bring up the fact that I don't get a lot of sleep, it turns into some stupid discussion with my health as the hot topic.

"You should try to get more sleep" or "have you tried sleeping pills?" are my favourite.

No, I can't just try to get sleep just like that. Yes, I have tried sleeping pills. About 40 of them at once when I tried to kill myself.

It wasn't fun.

Fig. 12: The character shares how he once tried to kill himself. Source: I’m Fine (2016)

3.1.8 Kiss Me

Kiss Me is a video game about a man who loves his girlfriend and tries to help her in her harshest moments of depression, anxiety and sadness (Sabiarts, 2017). The objective of the player is to paint the world with colorful colors in order to make the characters' girlfriend feel better (Fig.13). The game was developed by Sabiarts and released in 2017.

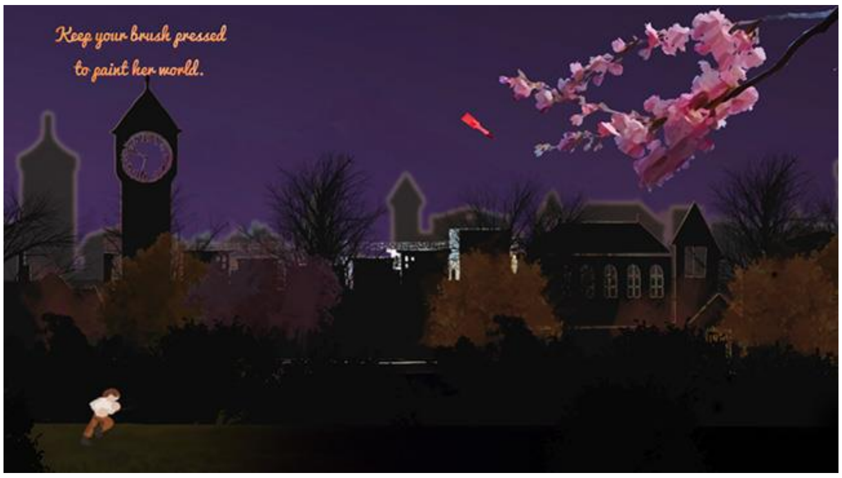
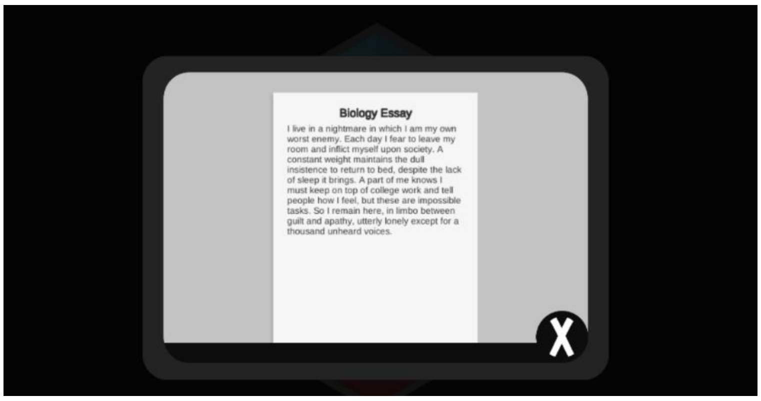


Fig. 13: Kiss Me Gameplay: the player has to paint the world of the characters' girlfriend in order to make her feel better. Source: Kiss Me (2017)

3.1.9 Get Help

Get Help is a video game about a depressed teenager who struggles with normal daily life tasks and chores (Akins *et al.*, 2018). The game was developed by William Akins, Ardhan Fadhlurrahman and Nicholas Clifton for the JAMMIND game jam, and it has been available to play since 2018. The player can help the character improve their life by doing a number of chores, but the character doesn't feel motivated nor energized enough to complete them. Instead of completing their chores and tasks, the character starts sharing their true feelings on the canvas of their essays (Fig.14).



Biology Essay

I live in a nightmare in which I am my own worst enemy. Each day I fear to leave my room and inflict myself upon society. A constant weight maintains the dull insistence to return to bed, despite the lack of sleep it brings. A part of me knows I must keep on top of college work and tell people how I feel, but these are impossible tasks. So I remain here, in limbo between guilt and apathy, utterly lonely except for a thousand unheard voices.

Fig. 14: The character shares their thoughts on their computer. Source: Get Help (2018)

Nothing feels like a good enough reason to try hard enough, in the character's mind. Nothing is really worth their effort, because when you are depressed, nothing really matters. As such, depression is represented through this system.

Chapter 4

The Development of Inside: A video game about Depression

This project consists of a video game that was developed using the Unity Engine, DragonBones, Adobe Photoshop, Adobe Premiere and SoundTrap.

4.1 Tools Used

4.1.1 Unity

Inside: A video game about Depression was developed in Unity. As a recent video game developer, diving in this practice means the choice between three main engines: Unreal, Unity or GameMaker. One can even consider Godot. However, Unity is a beginner friendly engine that allows developers to build games with the help of well founded documentation, flexibility and extensibility. There are plenty of successful games built with this engine, such as Hollow Night (Team Cherry, 2017), Overcooked (Team17, 2016) and sequel Overcooked 2 (Team17, 2018), Monument Valley (ustwo studio Ltd., 2014), and This War of Mine (11 bit studios, 2014). One cannot forget Unity is the Engine which is taught in the Game Design and Development Master's Degree, and this familiarity was an important point in the decision making process.

4.1.2 Game Objects

Game Objects are the core building block of Unity's Engine. Every single element one creates inside Unity must be wrapped in a Game Object - from cameras, effects, players, UI elements, and much more. Game Objects work such as a container. The logic behind nesting game objects in Unity lends a powerful tool with which developers can build software in a more dynamic and faster way.

4.1.3 Components

Game Objects on their own cannot do much. As said above, they are nothing more than mere containers. The power comes in when we add functionality to them and add components, which are, essentially, scripts written in either C Sharp or JavaScript. In this game, the choice was made to write all the logic and functionality using C Sharp. With the help of this powerful Object Oriented Programming Language, one can create several reusable components and join them into a large build. The level of reusability and clear communication between elements allows the decrease of unintended side effects such as small bugs that can spiral out of control. The main advantage of working with small building blocks and bringing them together in a large assembly is the control one has over the project.

4.1.4 Built-in Components

Unity has an array of important components that are pre-built in its system, which allowed the development of this game to happen in a much more simplistic and powerful way, such as:

- Camera: Allows the player to see the Game Object which is attached to it. In this game, Cameras are used in order to follow the main character through her journey and to create cutscenes that inform the player about the game.
- RigidBody: Determines how a game object moves and reacts to collisions. It also enables the building of realistic physics simulation. In this game, this component is used in order to give the main character a lightweight feel to her, as she jumps and flies around the game.
- Colliders: Allow the detection of Game Objects during collisions. This component was used in order to trigger many cutscenes and text that informs the player about the game.
- UI Canvas: Enables the creation of menus such as the main menu, health bars and score keepers. In this game, this component can be found in the various menus that are available in the game, as well as the health system of the main character.
- Light: Illuminates fractions of the scene, allowing the developers to give emphasis to certain areas, while making others darker.

There are many more components to be accessed, however, they weren't all used in the development of this game, and as such, talking about them is not needed.

4.1.5 Custom Components

The built-in components present in the Engine are very powerful. However, the game cannot be built solely with the use of those. Each game is different and in order to achieve that, developers have to create custom components with which they can create their own experience. In order to do this, one has to write C Sharp code. This Object Oriented Language allows the transformation of simple game objects into life filled environments and characters, each with its unique personality and flare.

4.1.6 User Input

Building a game that responds to user input in a clean and interactive way is quite simple using Unity. In order to achieve this, one has to simply use the engine's methods for grabbing user input, such as "Input.GetKey" or "Input.GetKeyDown".

4.1.7 Manipulating Game Objects

As we finally have access to User Input, we want our scene to respond to it. There are several ways through which a developer can achieve this, such as: translating, rotating or scaling an object, creating new Game Objects or sending messages to existing Game Objects and Components. In this game, the main character can jump, run and interact with the world she finds herself in due to this manipulation.

4.1.8 Animation

Apart from the animations related to the main character, every single animation that can be found inside this game has been made according to the engine's Animation System (Unity, 2020) - from the flowers and platforms that have animations that can be triggered when collisions are detected, to small enemies that patrol some game areas. Various states are managed with the help of the Animator System of the engine, which allows developers to change between states of the game and its different animations. Apart from these, the final cutscene, the one that happens after the end of the final boss fight, was also developed with the help of the Animation System provided by Unity.

4.1.9 Dragon Bones

In this bit, we will try to explain the reasons behind the preference of this software in detriment of the Animation and Bone System that can be found inside Unity. Dragon Bones is a Free 2D Skeletal Animation Software that allows developers to connect animations and game programming pipelines (Dragon Bones, 2017). This software allows users to create and bind bone systems inside the characters and create several animations with the help of these. We found it easier to create smoother and more vivid actions and motion with the help of this software when comparing with the Animation System that is built inside Unity. This was the only reason that affected the preference of one over the other – preference.

4.1.10 Adobe Photoshop

Adobe Photoshop is a graphic design and image creation software that allows users to edit photos, compose digital paintings, animation and graphic design (Adobe Photoshop, 2020). The graphic art part of the game was created with the help of Photoshop. Every single art element that one can see inside the game was designed and sketched inside Photoshop. Apart from this, the cutscenes displayed in the game were all created inside this software, with the help of the frame to frame animation system.

4.1.11 Adobe Premiere

Adobe Premiere is a video editing software for cinema, TV and the web (Adobe Premiere, 2020). This software allowed the composition and editing of the initial cutscene. After the completion of the animation segments in Photoshop, we created a number of transitions such as crossfades and fade ins. The color of the cutscene was also edited in order to keep the tones and palette of the game. The sound effects related to the cutscenes were also inserted with the help of Premiere.

4.1.12 Soundtrap

Soundtrap allows non-professional musicians to create beautiful sounds and music with the help of an extensive collection of loops, beats and instruments (Soundtrap, 2020). The music that can be found in the game was created with this online software.

4.2. Developing Inside – A Deeper Look

4.2.1. First Drafts and Ideas

Diving deeper inside the design and development journey, we will inform how the game was thought and composed. From the first sketches and ideas, all the way into a finished, final product.

The first steps after firming an idea of what the game was were designing a world capable of expressing the meaning of the game, as well as characters that could serve this purpose. The main character and the main boss enemy that we can see in the current version of the game are quite different from the ones that were created in the first drafts (Fig.15 and 16). After some considerations, we found it better to rethink and redesign the main character (Fig.17), as well as the main boss enemy (Fig.18).

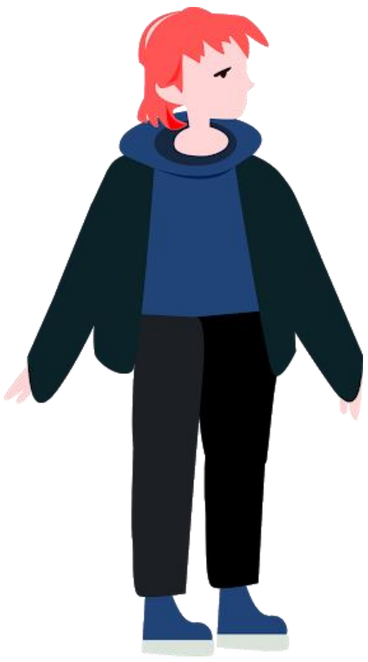


Fig. 15: First Draft of the main character.

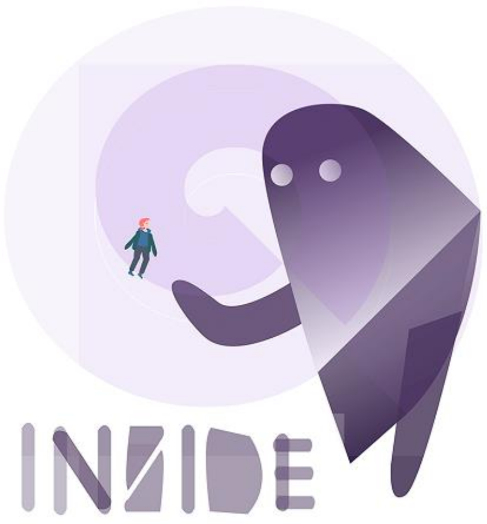


Fig. 16: First sketches and design of the main character (left) and depression (right).

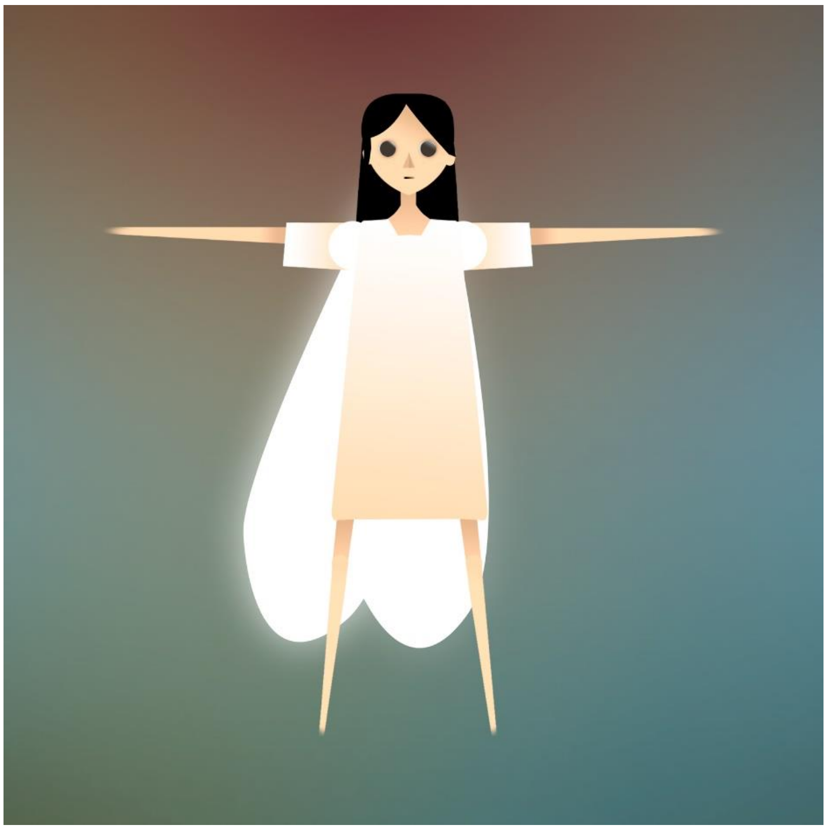


Fig. 17: Final version of the main character, Sirah.

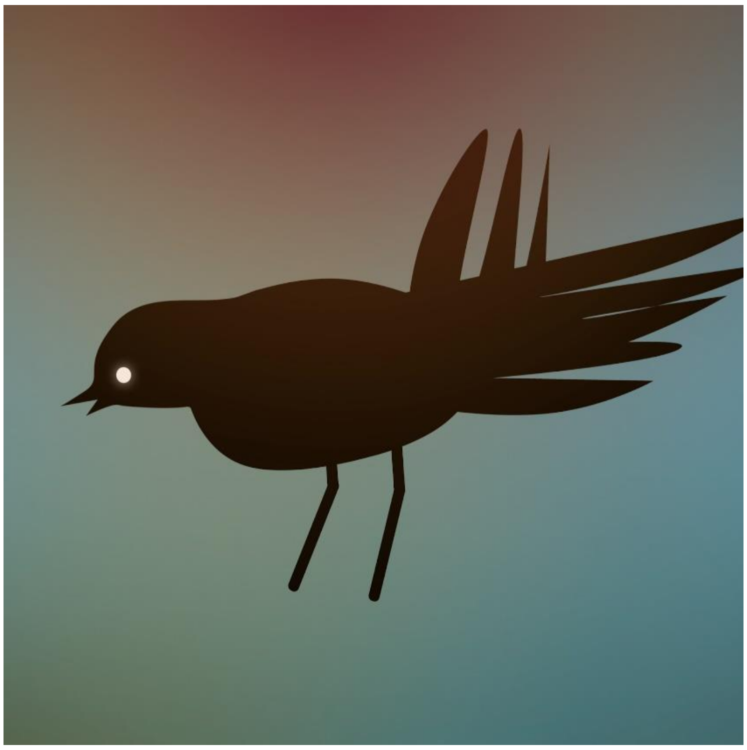


Fig. 18: Final version of the boss enemy, Depression.

The final boss fight was also thought to be different from what we ended up developing. In the first stages of development, the main character would be permanently followed by a looming figure, which would, in the end, try to squeeze it to death with its hands

(Fig.19). In the current and final version of the game, the boss enemy, which is depression, is represented by a black bird that flies after the main character, trying to detain her from leaving the coma (Fig.20).

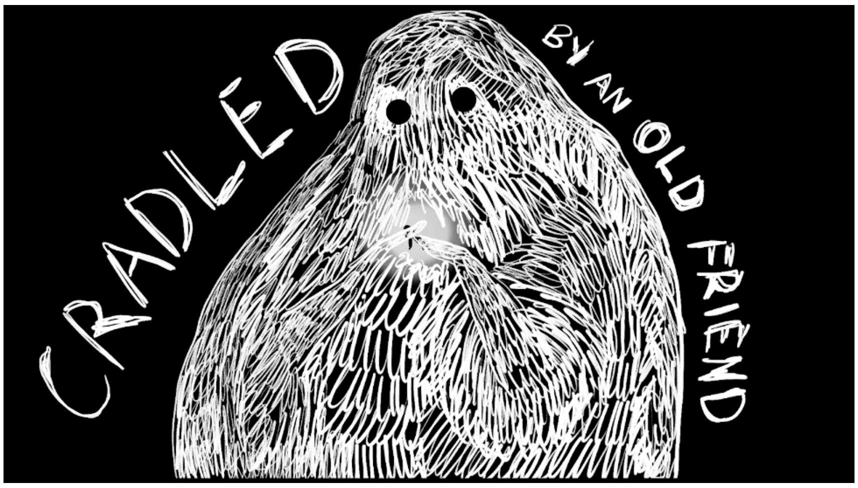


Fig. 19: First sketches and design of the boss fight.

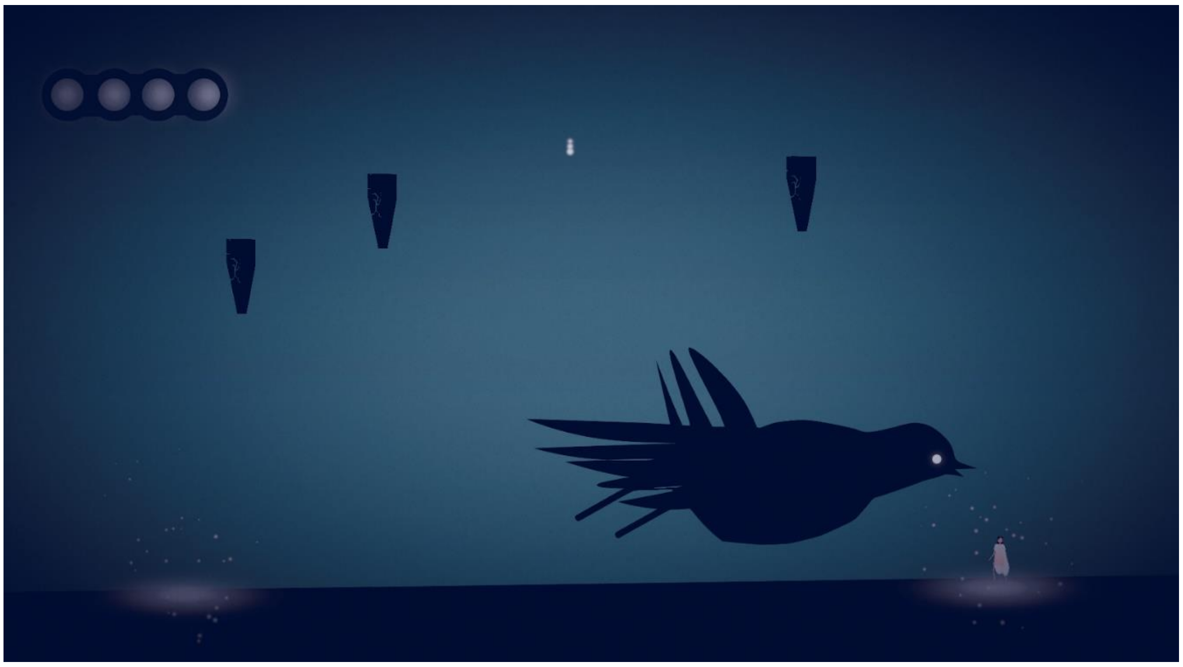


Fig. 20: Final and in-game version of the boss fight.

The game art cover was also thought to be different in the beginning. The first drafts consisted of an empty eyed girl looking into the distance, filled with a blue color (Fig.21), while the final version consists of the main character looking empty eyed as well but in a different, darker set (Fig.22).

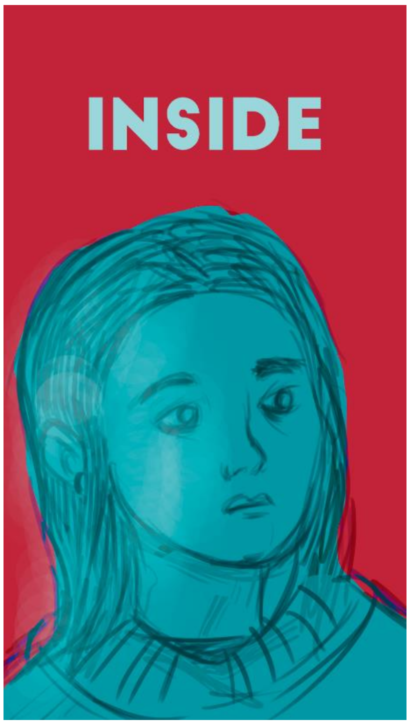


Fig.21: First draft of the cover art for Inside: A video game about depression.

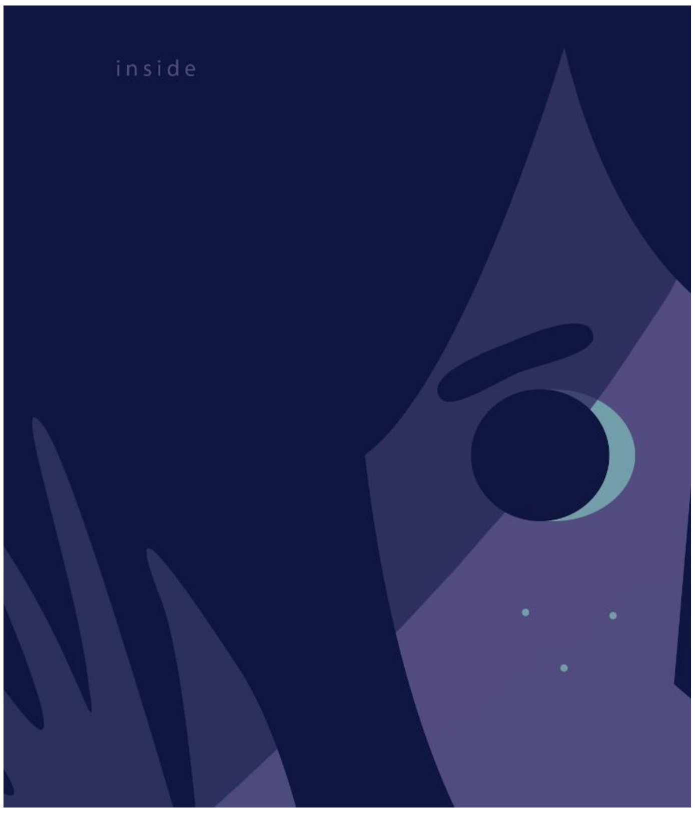


Fig.22: Final version of the cover art for Inside: A video game about depression.

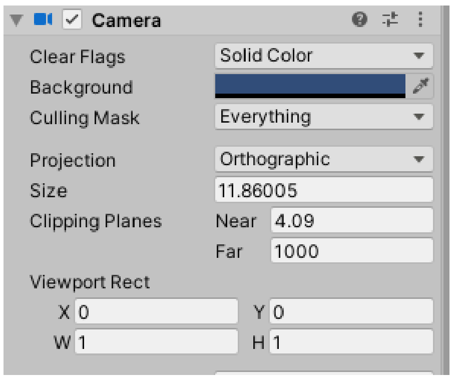
Many other elements, such as game mechanics or power ups and abilities, that appear in the first version of the Game Design Document (see Attachment 2), didn't make it into the final version of the game, or were transformed and used in different ways than the ones designed in the first sketches. Other systems and mechanics were present since the beginning and made it to the last version of the game.

In the next sections, we will talk about the various stages and steps related to the development of Inside**.** We will go through the process of creating the game feel, the building of the environment, the visual effects and every other component that is part of this game, such as the scripts used in order to achieve the intended objectives.

As stated above, the game was fully developed in Unity. As such, the first step towards the completion of this game was importing every visual asset created to Unity. The objective was to create a dynamic, stylized and somewhat realistic environment that allowed the player to feel like they were part of the world portrayed in the game. In order to achieve this, we applied a parallax effect in the game.

4.2.2 Parallax Effect

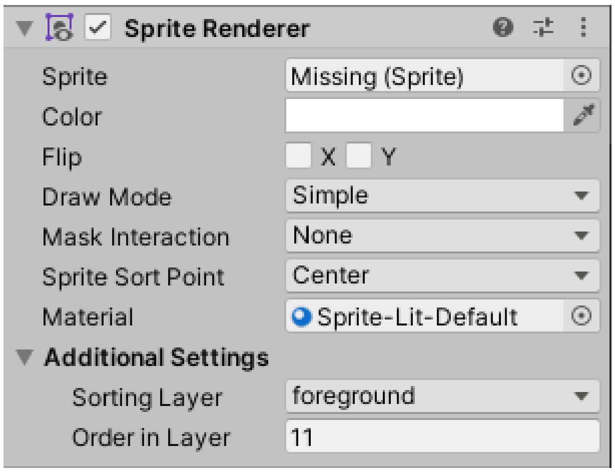
This effect is very important when working with a 2D video game. Parallax in games happens when the background moves slower than the elements that are closer to the camera, and it allows the creation of the illusion of depth. The first step towards including this effect in the game began with setting our camera component with an Orthographic Projection (Fig. 23). This is important because we are developing a 2D game, not 3D. If this game was to be 3D, setting the camera projection to Perspective would be enough to create the desired Parallax Effect in the game. Since we aren't doing so, we have to create our own Parallax Effect with the help of the manipulation of the camera component and the creation of some script files.


Camera

Clear Flags	Solid Color
Background	
Culling Mask	Everything
Projection	Orthographic
Size	11.86005
Clipping Planes	Near 4.09
	Far 1000
Viewport Rect	
X 0	Y 0
W 1	H 1

Fig.23: Unity's Camera Component with Projection set to Orthographic.

Another important step was setting the various visual assets in the right order. This is important because it allowed us to layer the game in a way that makes sense visually. In order to do this, we created different sorting layers, and defined different assets to be in different orders - for example, if we wanted an asset to be further in the background, then it would have a smaller order in layer than one that is closer to the camera (Fig. 24).



Sprite Renderer

Sprite	Missing (Sprite)
Color	
Flip	X Y
Draw Mode	Simple
Mask Interaction	None
Sprite Sort Point	Center
Material	Sprite-Lit-Default

Additional Settings

Sorting Layer	foreground
Order in Layer	11

Fig. 24: Unity's Sprite Renderer - Sorting Layer and Order in Layer.

The result of this work is the one presented in the figure placed under this portion of the text - an ordered environment that is familiar and makes sense to the player (Fig. 25).

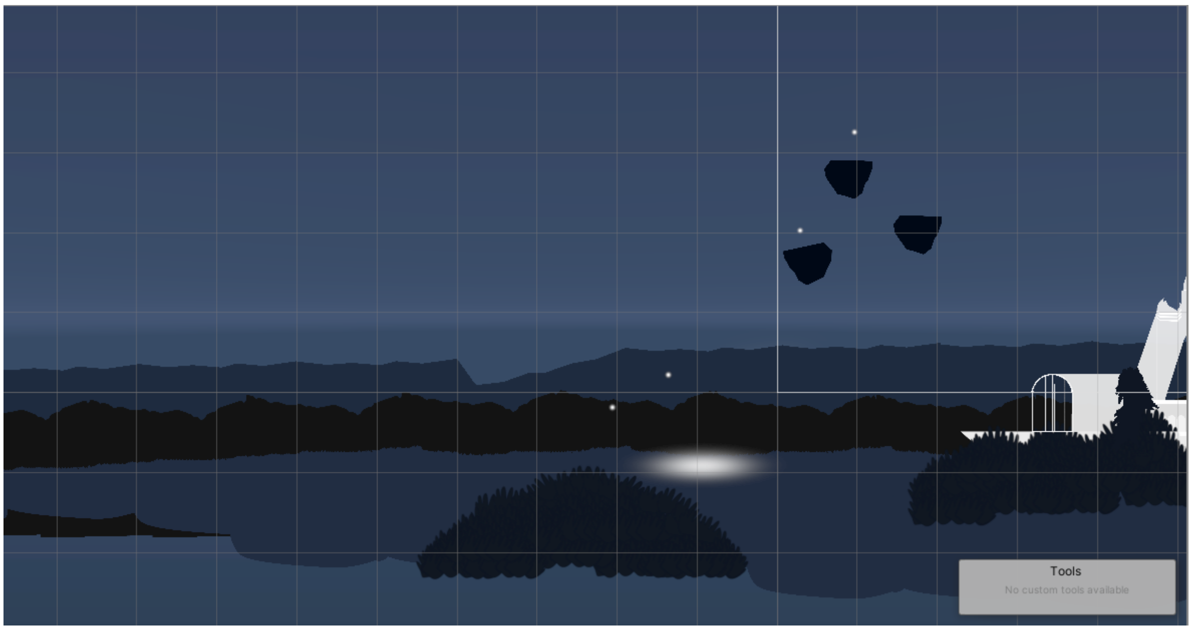
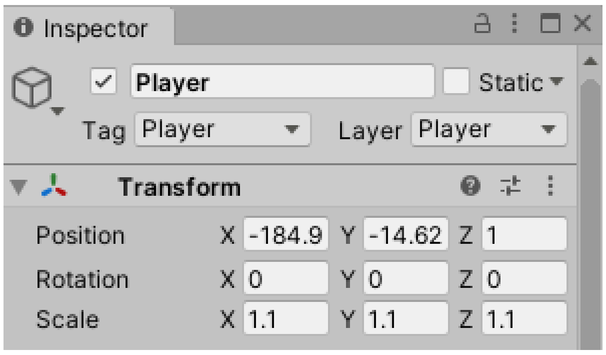


Fig.25: Unity's Scene View – The result of ordering layers.

Another important step is defining the z-axis for the different assets as a means to be able to create the Parallax Effect. As such, and as an example, the "sky" Game Object in this game is very far from the camera, so we set it to have a negative z-axis value, while an asset that is going to appear much closer to the camera, and, therefore, move faster, like the main character, is set to have a z-axis value of one (Fig.26). These values were set like this for the simple purpose of setting different images in different positions of the z-axis in order to organize all the layers appropriately (the images that are going to be further away in the background have a lower z-axis value, while the ones that appear closer have a higher value).


Inspector

Player — Static

Tag: Player — Layer: Player

Transform

Position	X -184.9	Y -14.62	Z 1
Rotation	X 0	Y 0	Z 0
Scale	X 1.1	Y 1.1	Z 1.1

Fig.26: Unity's Inspector - Player's z-axis set to the value of one.

The result of this work, viewed in a 3D view shows us that some components are indeed further, while others are closer to the camera (Fig.27). Our game camera is the small white rectangle that can be found on the right part of the image.

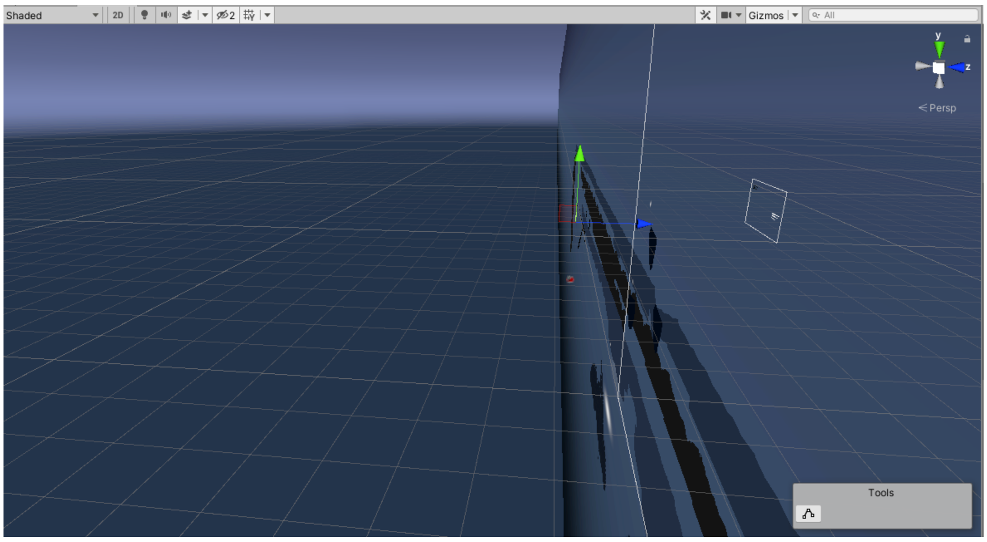


Fig.27: Unity's Scene View in 3D.

As everything was laid out as we wanted, the composition was set up for us to start scripting our 2D Parallax Effect. We need the scripts for both the Parallax Background, the Parallax Layer, as well as a script for the Parallax Camera itself, for reasons we will explain further. The Parallax Camera script was attached to our Main Camera. Furthermore, the code for the Parallax Background states that it is available to receive Parallax Layers (every single object that has a "Parallax Layer" attached to it) as their children, using them prior to this in order to move them in such a way that the parallax effect is created. This script is attached to te parent container of every game object that belongs to the game. Inside the Parallax Layer script, we can find the logic needed to create a two-dimensional parallax effect. We will show the code and explain how it works. The script is written as follows:

```
public class ParallaxLayer : MonoBehaviour
{
    public float parallaxFactor;
    public void Move(float delta)
    {
        Vector3 newPos = transform.localPosition;
        newPos.x -= delta * parallaxFactor;
        transform.localPosition = newPos;
     }
}
```

Listing 1: Parallax Layer Script

In order to create a 2D parallax effect, we need a variable of a parallax factor (public float parallaxFactor). This variable has to be public so that we can alter it as needed. After this, we created a "Move()" function in order to make the layer this script is attached to move around in relation to the parallaxFactor variable. The layers move with the parallax factor (public float parallaxFactor). This means that if we set this value to be "2", then it means that the layer affected by this script will always have a movement speed multiplied by this value, that is, the speed of this layer will be twice the camera's. If this value were to be multiplied by "0.5", then the layer would move two times slower than the camera. If we set it to "1", then the layer will move the same speed as the camera. This is quite interesting because we want the things that are furthest from the camera to move slower and the things that are closest to the camera to move faster. This said, we set the grass sprite to have a parallax factor of zero, while the sky has a value of "0.9". The bushes that are very close to the camera move quite fast, and for that, have a higher parallax factor (Fig.28).

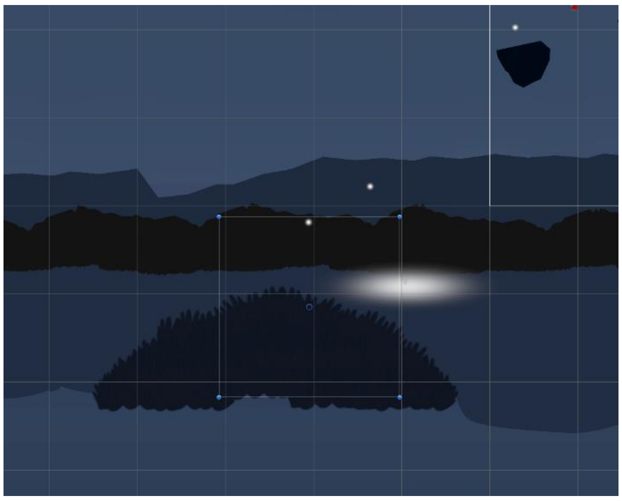
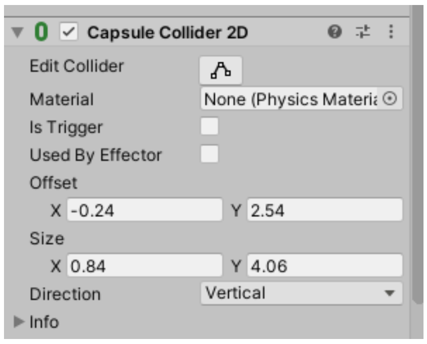


Fig.28: Scene's visual hierarchy.

The final result of this work is an environment that works and moves beautifully, bringing some recollection and visual familiarity to the player.

4.2.3 The Player (Game Object)

For our Player Game Object, we will need to create custom scripts in order to achieve our goals. Building the main character starts with applying a script that will help us deal with the movement of the player, such as jumping and running, the health system, which defines which collisions result in a gain or loss of health, as well other elements that will be discussed further down this dissertation. Furthermore, a Rigidbody 2D must be applied to the player as well, as we stated above when we were talking about the tools used in this game. Inside the Rigidbody 2D component, the Body Type was also set to Dynamic in detriment to Kinematic and Static. In order to detect collisions, we added a 2D capsule collider to the player game object (Fig.29).



Capsule Collider 2D
Edit Collider
Material None (Physics Materi:
Is Trigger
Used By Effector
Offset
X -0.24 Y 2.54
Size
X 0.84 Y 4.06
Direction Vertical
Info

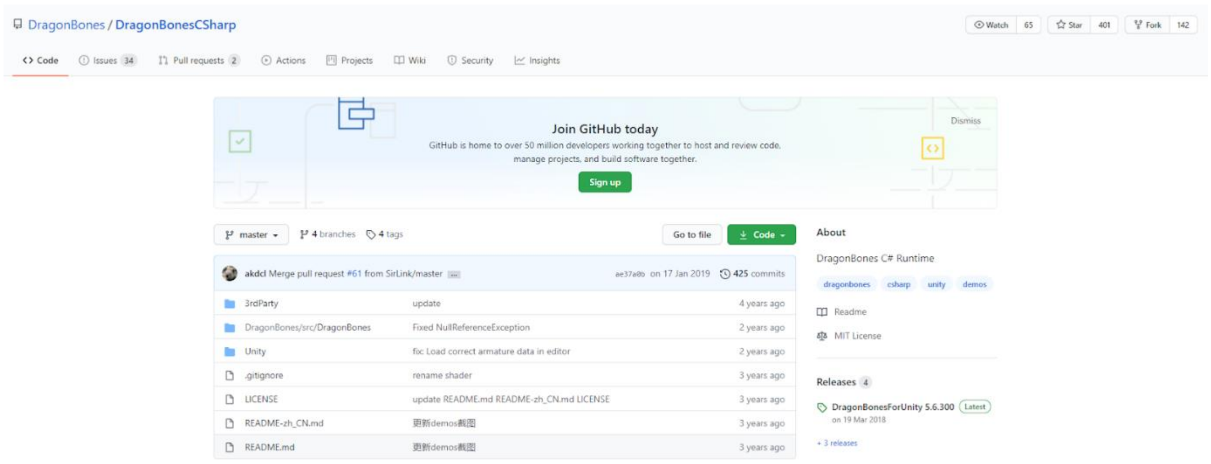
Fig.29: Unity's Inspector - Player's Components – Capsule Collider 2D.

The capsule was chosen instead of a box collider mainly because the square's sharp corners of the box collider dig inside other colliders, making the player's movement less realistic and smooth, causing, many times, the player to get stuck.

4.2.4 The Player 's Animations

As previously stated, the animations related to the player game object, our main character, also known as Sirah, were created with the 2D animation software Dragon Bones. Dragon Bones allows Photoshop (.psd) files to be directly imported to its software, and as such, after doing that, we had to create the bones for our character (Dragon Bones, 2017). The animations were developed with keyframes, and we managed to create three main ones: Running, Jumping and Idle. Exporting a Dragon Bones project to be used inside Unity starts with saving it as a ".dbproj" file. This process generated three files – a ".tex.png" file, a ".tex.json" file, and lastly, a "ske.sjon" file, that contains the skeleton and the animation information.

Importing Dragon Bones files into Unity and making them work with the help of some logic was a bit hard. The API documentation was, sometimes, unhelpful. However, we ended up figuring out that there is a Dragon Bones plugin for Unity's engine (GitHub, 2016) that can be found on GitHub (Fig.30).



Join GitHub today
GitHub is home to over 50 million developers working together to host and review code, manage projects, and build software together.

akdcl Merge pull request #61 from SirLink/master ae37a0b on 17 Jan 2019 425 commits

3rdParty	update	4 years ago
DragonBones/src/DragonBones	Fixed NullReferenceException	2 years ago
Unity	fix Load correct armature data in editor	2 years ago
.gitignore	rename shader	3 years ago
LICENSE	update README.md README-zh_CN.md LICENSE	3 years ago
README-zh_CN.md	更新demos截图	3 years ago
README.md	更新demos截图	3 years ago

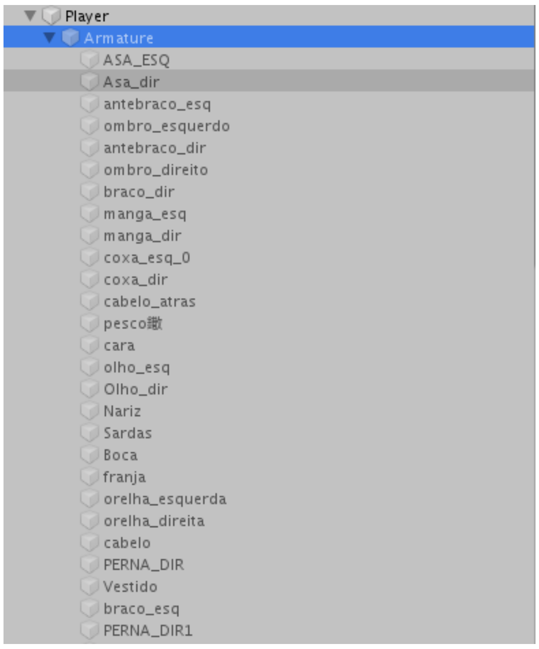
About
DragonBones C# Runtime
dragonbones csharp unity demos
Readme
MIT License

Releases 4
DragonBonesForUnity 5.6.300 Latest
on 19 Mar 2018
+ 3 releases

Fig. 30: Dragon Bones plugin. Source: Git Hub (2016)

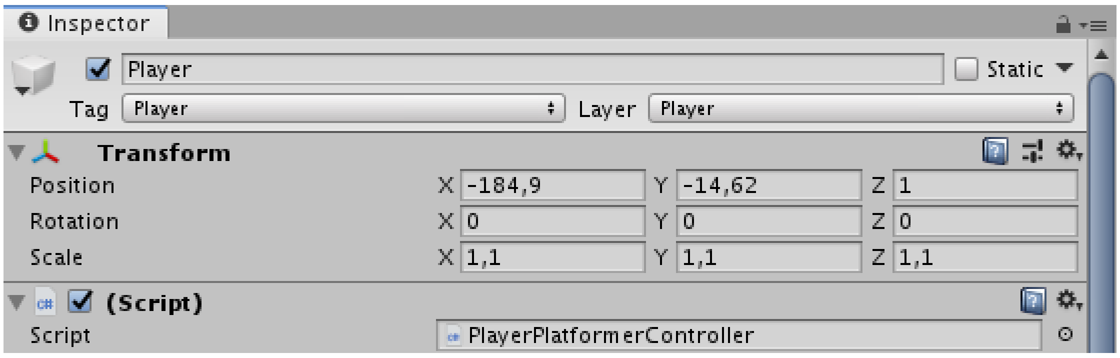
The plugin was imported inside our Unity project as a package. It helped us in the proccess of not only importing the Dragon Bones assets into our Unity Project, but also made it possible to work with such assets. With access to the Dragon Bones files, we then had the chance to create the Armature Object that contained all the animations of our main character (Fig.31). Inside this Armature Object, a Unity Armature Component was automatically created. This component could now be easily accessed via script, which meant we could work with all of the animations we had created previously inside Dragon

Bones. With this work done, we started implementing the functionality and behaviour of our main character's animations. As such, inside the Player game object that parents the Armature Object, we added our C Sharp code, ruled by the name of PlayerPlatformerController (Fig.32).



```
Player
  Armature
    ASA_ESQ
    Asa_dir
    antebraco_esq
    ombro_esquerdo
    antebraco_dir
    ombro_direito
    braco_dir
    manga_esq
    manga_dir
    coxa_esq_0
    coxa_dir
    cabelo_atras
    pesco鐵
    cara
    olho_esq
    Olho_dir
    Nariz
    Sardas
    Boca
    franja
    orelha_esquerda
    orelha_direita
    cabelo
    PERNA_DIR
    Vestido
    braco_esq
    PERNA_DIR1
```

Fig. 31: Armature Object of the Player Object.



```
Inspector
Player                                   Static
Tag  Player                 Layer  Player
Transform
Position        X -184,9   Y -14,62   Z 1
Rotation        X 0        Y 0        Z 0
Scale           X 1,1      Y 1,1      Z 1,1
(Script)
Script          PlayerPlatformerController
```

Fig. 32: Player Platformer Controller Component inside Player game object.

Inside this script, we had to state that we were using the DragonBones namespace in order to have access to the components we wanted to work with (Listing 2).

```
using DragonBones;
```

Listing 2: Dragon Bones Unity namespace.

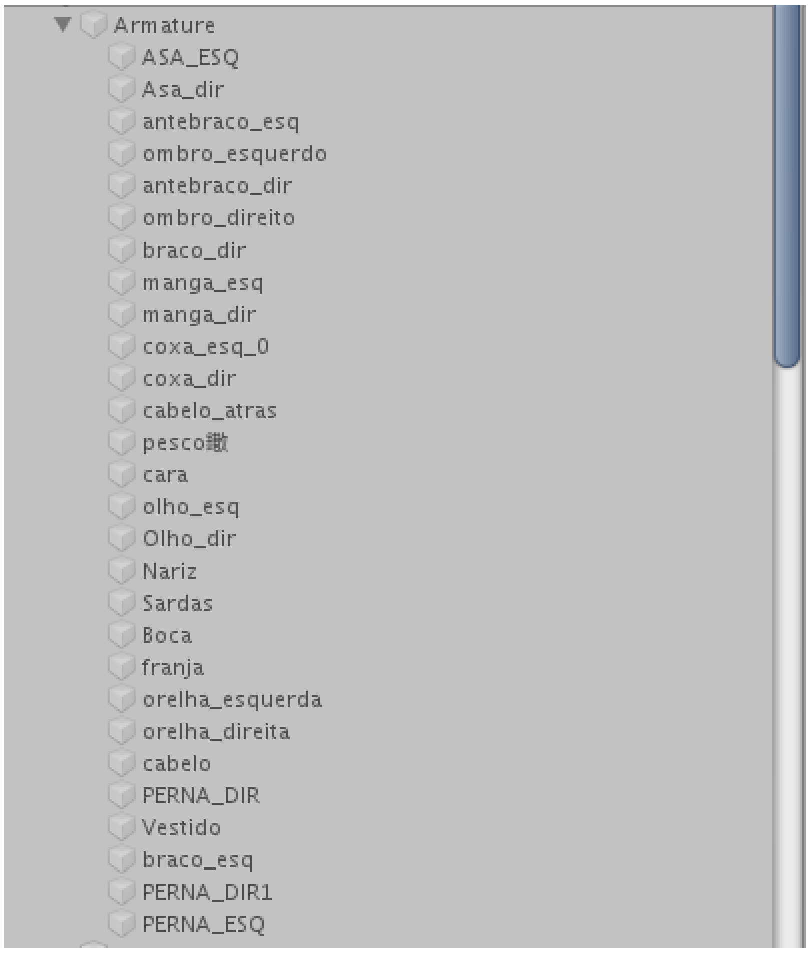
With this completed, we created a public variable for our Unity Armature component called anim (Listing 3), and referenced it inside our Start() function, stating that it would fetch every child inside this component (Listing 4). This means that every sprite that is part of the whole that is the armature component would be fetched in order to be used (Fig.32).

```
public UnityArmatureComponent anim;
```

Listing 3: UnityArmatureComponent public variable “anim”.

```
  void Start()
    {
        anim = 
transform.GetComponentInChildren<UnityArmatureComponent>();
    }
```

Listing 4: Code for getting every child inside the armature component.



```
Armature
    ASA_ESQ
    Asa_dir
    antebraco_esq
    ombro_esquerdo
    antebraco_dir
    ombro_direito
    braco_dir
    manga_esq
    manga_dir
    coxa_esq_0
    coxa_dir
    cabelo_atras
    pesco鐵
    cara
    olho_esq
    Olho_dir
    Nariz
    Sardas
    Boca
    franja
    orelha_esquerda
    orelha_direita
    cabelo
    PERNA_DIR
    Vestido
    braco_esq
    PERNA_DIR1
    PERNA_ESQ
```

Fig. 32: Armature Object and its’ children (sprites used to create the animations inside Dragon Bones).

Inside the "Start()" function, we also stated that the default animation that would play until another event triggered a different behaviour would be the "stand" animation, an idle animation that we created inside Dragon Bones (Listing 5).

```
void Start()
    {
        anim.animation.Play("stand");
    }
```

Listing 5: Code for playing the default animation, "stand".

Otherwise, if we detected an input that affected the position of the character, we had to connect each animation to said movement. This said, we then applied this logic both inside our "FixedUpdate()" function, where we defined how we wanted to read the player's input, as well as a function created by us named "ChangeAnimation()", that read and ordered how animations changed. In order to do this, we had to start by creating a variable named moveInput (Listing 6) and work around it.

```
protected float moveInput;
```

Listing 6: Variable for the move input

With this step concluded, we then defined this variable to get the player's input inside our "FixedUpdate()" function, more specifically, the horizontal axis, that reads if a player is trying to either move left or right (Listing 7).

```
float moveInput = Input.GetAxisRaw("Horizontal");
```

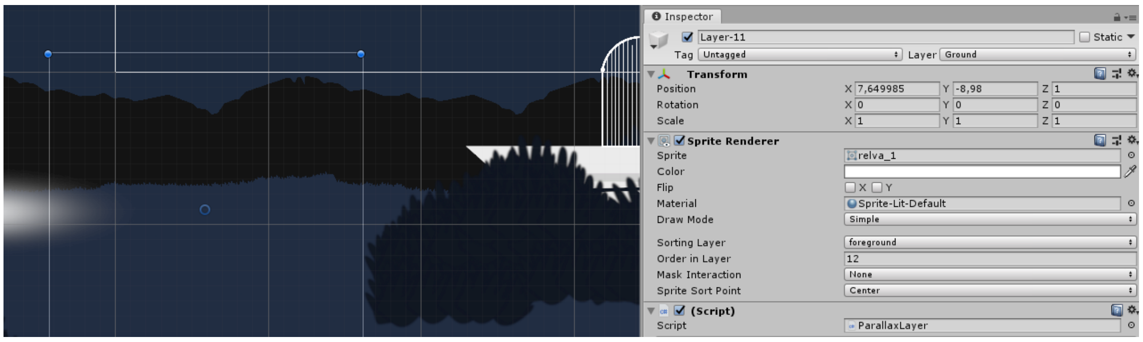
Listing 7: Code for reading the player's input.

Apart from this, we also needed to know if our player object was indeed touching the floor or not (Listing 8). If they were, then the player would be grounded. If not, then they would not. We had to create this variable in order to be able to create a double jump.

```
isGrounded = Physics2D.OverlapCircle(groundCheck.position,
checkRadius, whatIsGround);
```

Listing 8: Variable that detects if the player is touching the floor collider or not.

The way we managed to detect if the Player object was touching the floor collider or not was through creating a circle collider positioned on the character's feet. This way, "groundCheck.position" stands for the position of the circle collider, "checkRadius" stands for the radius of this circle collider, and "whatIsGround" stands for the name of the layer (Fig.33) we applied to every single sprite which is, indeed, part of the floor (Listing 9). This means that this variable will check if the sprites the player object is touching with her feet are indeed called "Ground".



```
[✓] Layer-11                         [ ] Static
Tag  Untagged           Layer  Ground
Transform
Position            X 7,649985   Y -8,98   Z 1
Rotation            X 0          Y 0       Z 0
Scale               X 1          Y 1       Z 1
[✓] Sprite Renderer
Sprite              relva_1
Color
Flip                [ ] X [ ] Y
Material            Sprite-Lit-Default
Draw Mode           Simple
Sorting Layer       foreground
Order in Layer      12
Mask Interaction    None
Sprite Sort Point   Center
[✓] (Script)
Script              ParallaxLayer
```

Fig. 33: The grass sprite that is selected has a Layer named "Ground".

```
public LayerMask whatIsGround;
```

Listing 9: Variable for detecting the ground Layers.

This means that, every time the circle collider entered a layer named "Ground" then the player would be grounded (the Boolean "isGrounded" would be true).

With this work done, we then started building our "ChangeAnimation()" function. We started by defining that the function would take in a parameter, the "moveInput" variable. After this, we created several cases for our animations. If the Player object was touching the floor, and if the "moveInput" variable was taking in a value of "-1.0f", then the Player object would be playing the running animation towards the left (Listing 10).

```
void ChangeAnimation(float moveInput){
        switch(moveInput)
        {
            case -1.0f:
                if(isGrounded){
                    anim.animation.FadeIn("Correr", 0.01f, -1);
                }
                break;
        }
    }
```

Listing 10: Script for running to the left.

Otherwise, if the Player object was receiving a “moveInput” value of “1.0f”, then the player would be running towards the right (Listing 11).

```
void ChangeAnimation(float moveInput){
        switch(moveInput)
        {
          case 1.0f:
            default:
                if(isGrounded){
                    anim.animation.FadeIn("Correr", 0.01f, -1);
                }
            break;
        }
    }
```

Listing 11: Script for running towards the right.

Contrarily to the previous two cases, if there was no move input at all, then the player would not move, and the idle animation (“stand”) would play (Listing 12).

```
void ChangeAnimation(float moveInput){
        switch(moveInput)
        {
          case 0.0f:
                if(isGrounded){
                    anim.animation.FadeIn("stand", 0.01f, -1);
                }
                break;
        }
    }
```

Listing 12: Script for playing the idle animation.

However, this resulted in the sprite always being flipped to the right, which resulted in the player running right and left looking always to the right. In order to fix this, we had to make the Player object flip according to the input axis. Hereinafter, we created both a boolean variable named "facingRight" that was set to true by default (Listing 13) and a function named "Flip()" (Listing 14).

```
    private bool facingRight = true;
```

Listing 13: Variable to check if the Player object is facing the right or the left.

```
void Flip()
    {
        facingRight = !facingRight;
        Vector3 Scaler = transform.localScale;
        Scaler.x *= -1;
        transform.localScale = Scaler;
    }
```

Listing 14: Function to flip the Player object according to the x axis.

With this function, we made it so that if the current state was "facingRight" (facing the right) then it would change into "!facingRight" (facing the right would change into facing the left) and so on. Following this step, we had to insert this logic in our script concerning the animations. Consequently, and returning to our "ChangeAnimation()" function, if the player was facing right but the move input returned a value of "-1.0f", then the player would flip, this is, turn left, and if the player was facing left and the move input returned a value of "1.0f", then the player would flip again to the right (Listing 15).

```
    void ChangeAnimation(float moveInput){
        switch(moveInput )
        {
            case -1.0f:
                if(facingRight){
                    Flip(); //moving Right
                }
                if(isGrounded){
                    anim.animation.FadeIn("Correr", 0.01f, -1);
                }
                break;
            case 0.0f:
                if(isGrounded){
                    anim.animation.FadeIn("stand", 0.01f, -1);
                }
                break;
            case 1.0f:
            default:
                if(!facingRight){
                    Flip(); //moving Right
                }
                if(isGrounded){
                    anim.animation.FadeIn("Correr", 0.01f, -1);
                }
            break;
        }
    }
```

Listing 15: The Flip() function integrated inside the ChangeAnimation() function.

After the completion of this step, the only animation left to be connected to our character was the jump animation. As a means to achieve this, we declared inside the "FixedUpdate()" function that if the Player object was grounded and if the jump animation wasn't being played, two jumps could be used. If one jump had already been used, only one would be left, and if both were already used, then the player Object would not have any more jumps available and would fall. This way, the player would have the total of two jumps every time they left the ground (Listing 16).

```
if(!isGrounded && !isPlayingJump){
      anim.animation.FadeIn("saltar",0.01f, -1);
      isPlayingJump = true;
      }
else if(isGrounded && isPlayingJump){
      isPlayingJump = false;
      anim.animation.Stop("saltar");
      ChangeAnimation(moveInput);
```

Listing 16: Code for making the player jump and play the jump animation.

This way, we created the funcionality of a double jump, so that the player could jump higher if intended. On top of that, we also wanted the player to have the capacity of charging the jump in order to jump higher. As a means to accomplish this, we had to create multiple variables (Listing 17).

```
public int extraJumps; //how many extra jumps are available
public float jumpForce; //how high will the player go
```

Listing 17: Variables used developing the double jump mechanic.

Then, in the "Update()" function of our PlayerPlatformerController script, we created the logic necessary to allow this to happen. As a mean to achieve this, we had to check if there was na input from the player, and if this input was being triggered by the spacebar, and if the extra jumps hadn't been used yet, then the player would be able to jump and double jump. Contrary to this, if the player had already jumped once, they would be able to do it only once again (Listing 18).

```
if (Input.GetKeyDown(KeyCode.Space) && extraJumps > 0) {
                StartCoroutine(StartCounting());
        } else if (Input.GetKeyDown(KeyCode.Space) && extraJumps ==
0 && isGrounded == true) {
            rb.velocity = Vector2.up * jumpForce;
        }
if(Input.GetKeyUp(KeyCode.Space) && extraJumps > 0){
            StopCoroutine(StartCounting());
            rb.velocity = Vector2.up * jumpForce;
            jumpForce = 9;
            extraJumps--;
        }
```

Listing 18: Code used for making the Player object double jump.

As we can see, we are also using a Coroutine. This coroutine is related to the "press longer in order to jump higher" system we integrated in the game. Thus, the coroutine we created, named "StartCounting()", needed to assess how long the player would be pressing the space bar and when they released it (Listing 19).

```
IEnumerator StartCounting()
        {
            for (holdTime = 0.1f; holdTime <= 5f; holdTime +=
Time.deltaTime*2) {
                if(jumpForce <= 30){
                    jumpForce = jumpForce+holdTime;
                    yield      return      new      WaitForSeconds
(Time.deltaTime);
                }
            }
            holdTime = 1f;
            jumpForce = jumpForce+holdTime;
        }
```

Listing 19: Code used to create an algorithm that would allow the player to press the space bar longer in order to jump higher.

This behaviour, connected to the double jump mechanic applied to the Player object allowed us to give a light feel to the main character. Seen that we had completed every movement and motion aspect of our character, we could start working on the other objects the Player object would interact with.

4.2.5 Polygon Colliders

2D Polygon Colliders allow the player to interact with the environment that surrounds them. As a first step, it is important to set a collider on the floor, so that the player can be grounded and then walk, jump, and develop every motion. The collider, in this case, is not set as a trigger because we want the player to walk over it. The 2D Polygon Collider differs from the 2D Box Collider in a way that the first is editable, vertices wise, while the second isn't. What we mean by this is that while a box can be a square or a rectangle, the 2D Polygon Collider can take any shape we want it to be. This is advantageous because we can create a more realistic and smoother feel to the ground, as our player walks around it (Fig.34). In this game, our collider set on the floor was placed a bit under the art asset of the grass, mainly because we wanted to portray the player as walking inside the grass, not on top of it.

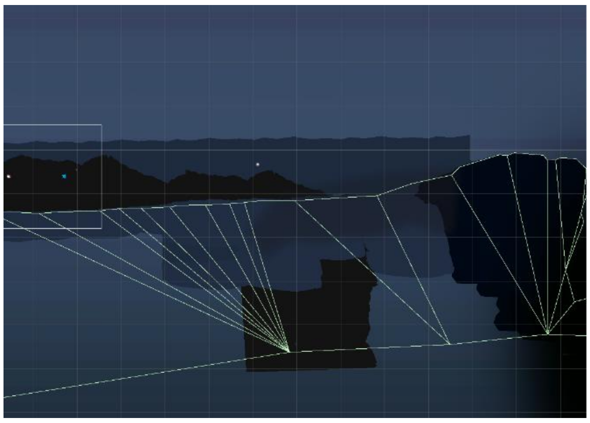
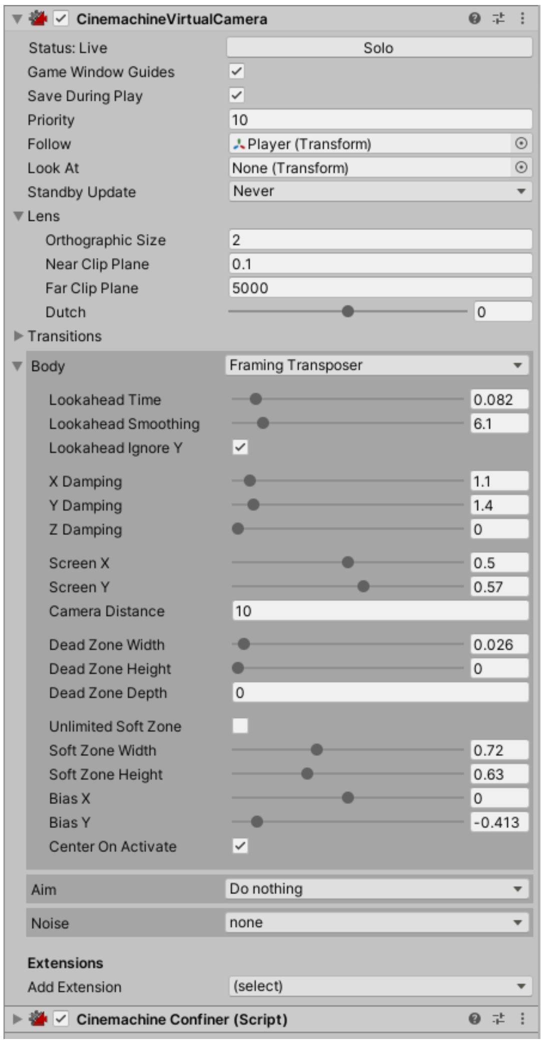


Fig. 34: 2D Polygon Collider: Floor collider.

4.2.6 Cinemachine

In this game, the camera follows our player through their path. Most of the times, the camera focuses only the player, but other times, we had to make the camera follow other platforms and objects with the help of cutscenes. Cinemachine is a great tool that helped us do that. As such, there are different ways to proceed in order to make both cases happen. In order to make the camera follow the Player object, we had to create a new 2D Camera attached to the Main Camera object with the help of the Cinemachine package. Inside this new camera, a Cinemachine Virtual Camera script was created automatically after the previous step. We didn’t change any code, we simply set the camera to follow our player (Fig.35). This way, as the player moves, the camera moves as well, instead of the camera being static and the player moving, resulting in Player object leaving the screen visible to the user.



CinemachineVirtualCamera	✓
Status: Live	Solo
Game Window Guides	✓
Save During Play	✓
Priority	10
Follow	Player (Transform)
Look At	None (Transform)
Standby Update	Never
Lens	
Orthographic Size	2
Near Clip Plane	0.1
Far Clip Plane	5000
Dutch	0
Transitions	
Body	Framing Transposer
Lookahead Time	0.082
Lookahead Smoothing	6.1
Lookahead Ignore Y	✓
X Damping	1.1
Y Damping	1.4
Z Damping	0
Screen X	0.5
Screen Y	0.57
Camera Distance	10
Dead Zone Width	0.026
Dead Zone Height	0
Dead Zone Depth	0
Unlimited Soft Zone	☐
Soft Zone Width	0.72
Soft Zone Height	0.63
Bias X	0
Bias Y	-0.413
Center On Activate	✓
Aim	Do nothing
Noise	none
Extensions	
Add Extension	(select)
Cinemachine Confiner (Script)	✓

Fig. 35: Cinemachine Virtual Camera set to follow Player object.

The vantageous point of Cinemachine is that we can create multiple virtual camera game objects that are activated and deactivated and control only one camera. What this means is that, instead of having 5 or 6 cameras with different traits, we can have 5 or 6 game objects that we activate and deactivate according to our will. A good example of the use of this system in our game is the creation of cutscenes every time the player collides with a certain collider that triggers it. We have set one main camera that follows our player, as stated above, but we also have three other cameras that follow some platforms, inform the player how to cross a certain level and even expand the screen during the Boss Fight.

4.2.7 Building a world and populating it

Creating an environment the player can travel through was an important part of developing a storytelling game. The spaces where our main character travels through inform the player about what is happening and the other living objects that interact with them are essential when trying to create an immersive game. As such, we tried to design a whole ambient that would quickly make the player understand what the game is about. In order to achieve this, we had to create dark sprites, ominous sounds and windy animations. We wanted the set to be as empty as possible, in order to share a feeling of loneliness. We also placed some fireflies and some orbs the player can catch and other enemies in the game environment. Apart from this, there is a continuous monologue that is shared in the form of text that appears in the screen. As a whole, the game tries to portray the loneliness and mental health of the main character, Sirah, through various elements.

We will develop this section further as we explain how the orbs, the monologues, the small enemies and other elements work in accordance to creating a world for our character.

4.2.8 Orbs and Health

As we know, the main character suffers from Depression and tried to commit suicide. With this in mind, creating shiny, soft, white orbs seemed to work well in contrast to the darkness of the environment that the character finds herself in (Fig.36). Through the perspective of game design, in this game, we can think that the bright, light elements are good, while the dark ones are bad. The enemies and the whole set where Sirah runs through are very dark, while the orbs, the fireflies and the puzzles that allow the player to get closer to the end of the game are very bright. We created this duality with the aim of making things understandable from the point of view of someone who is playing and experiencing the story of the game for the first time.

In this game, the player can catch the orbs with the aim of either being able to unlock puzzles or not diying when confronting the Boss Enemy. In this section, we will display how we created such systems and how they helped us build the game story.

We created a score system that also serves the purpose of a health system that can be found above the Player object, on the left top corner of the screen, that informs whoever is playing the game about how many orbs one can have, has, or has lost (Fig.37).

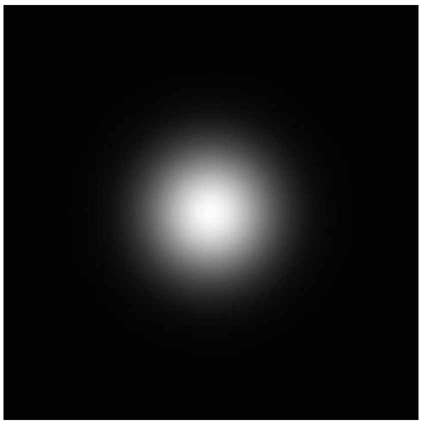


Fig. 36: Orb Sprite.

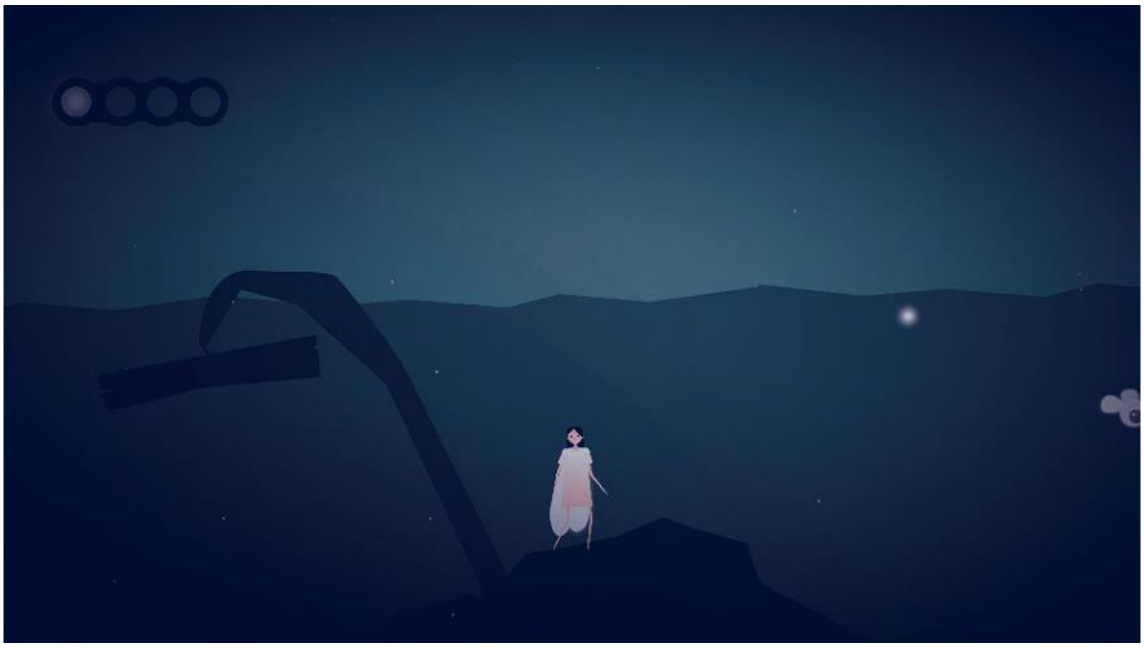


Fig. 37: Score System set on the top left corner of the game view.

Every time the player catches an orb, the score system is filled up with an orb. On the contrary, if the player collides against an enemy, they lose an orb from the score system. In an effort to accomplish this, we had to create the logic for the score system, connect it to the health system of the Player object and declare which objects could decrease or increase the number of orbs. In our case, the collisions that resulted in the increase of the number of orbs were, indeed, the collisions with the orbs that are navigating around the world, while the collisions responsible for the decrease of the number of the orbs were the ones that happened against the enemies. In order to achieve this, we created and associated two scripts to the score system object, one responsible for the visual part of the system, the other, the behaviour and logic. Inside the script related to the visual part of the score system, we had to create two variables for the sprites that would fill up the score system. The first one is associated to the sprite that appears every time the orb fills up one step, while the second variable concerns the sprite that appears every time the score system loses one orb (Listing 20).

```
private Sprite orbSprite;
private Sprite orbSpriteEmpty;
```

Listing 20: Score System Orb sprites.

In regards to the score system having more than one orb being added or subtracted, we also had to create a variable that would contain a list of orb images (Listing 21).

```
private List<OrbImage> orbImageList;
```

Listing 21: Variable for the Orb list.

We also defined that the default count of the available orbs inside the score system would be zero (Listing 22). We chose to do this because we wanted the player to interact with an orb and understand its power as soon as possible.

```
public int activeOrbs = 0;
```

Listing 22: Variable for the active orbs inside the score system.

With these steps out of the way, we started developing our score system functionality. However, before we dive into this subject, we have to explain why we built our code the way we did. The functions “Awake()” and “Start()” are both called automatically when a script is loaded. However, “Awake()” is always called first, even if the script component is not enabled, and is best used for setting up references between scripts and initialization. On the other hand, “Start()” is called after the “Awake()” function, immediately after the first update, but only if the script component is enabled (Unity, 2020). Since we needed a new list of orb images to be created every time the Player object entered a new scene, we chose the “Awake()” function to render them. On the “Awake()”

function, we created a new orb image list, so that we could access it immediatley inside the game (Listing 23).

```
private void Awake()
    {
        orbImageList = new List<OrbImage>();
    }
```

Listing 23: A new list of orb images is created on the Awake() function.

On our “Start()” function we received the logic from the second script related to the score system, the Orb Bar System script. Here, we defined that the score system would take only four orbs (Listing 24). We used the “Start()” function to carry the logic of this system because inside it we can find the actual content that is going to fill the list we are initializing inside our “Awake()” function. In a certain way, we can affirm that the “Awake()” function was working as a placeholder for the system the “Start()” function would fill it with.

```
private void Start()
    {
      OrbBarSystem orbBarSystem = new OrbBarSystem(4);
      SetOrbCounterSystem(orbBarSystem);
    }
```

Listing 24: Creating a new orb bar system with the value of four inside the Start() function.

With this done, we then created a couple of functions responsible for running through every orb, adding or subtracting orbs and refreshing the image orb list, depending on what was happening (“SetOrbCounterSystem()” and “RefreshAllOrbs()”) (Listing 25). If the Player object was receiving damage, then the score system would lose one orb. On the other hand, if the Player object was catching an orb, then the list would gain one orb. Aditionally, every time the Player object crossed a puzzle and went to the next scene, the Orb list, the counter system, was reseted (“RefreshAllOrbs()”).

```
public void SetOrbCounterSystem(OrbBarSystem orbBarSystem)
    {
        this.orbBarSystem = orbBarSystem;
        orbsHealthSystemStatic = orbBarSystem;

        List<OrbBarSystem.Orb> orbList = orbBarSystem.GetOrbList();
        Vector2 orbAnchoredPosition = new Vector2(-836.5f, 370);
        for (int i = 0; i < orbList.Count; i++)
        {
            OrbBarSystem.Orb orb = orbList[i];

CreateOrbImage(orbAnchoredPosition).SetOrbFragments(orbList.Count);
            orbAnchoredPosition += new Vector2(+75.5f, 0);
        }
        orbList.RemoveRange(0, orbList.Count);
        RefreshAllOrbs();
        orbBarSystem.OnDamaged += OrbBarSystem_OnEvent;
        orbBarSystem.OnHealed += OrbBarSystem_OnEvent;
        orbBarSystem.OnDead += OrbBarSystem_OnEvent;
        orbBarSystem.OnRestart += OrbBarSystem_OnEvent;
    }

private void RefreshAllOrbs() {
        List<OrbBarSystem.Orb> orbList = orbBarSystem.GetOrbList();
        activeOrbs = orbImageList.Count;
        for (int i = 0; i < activeOrbs; i++)
        {
            OrbImage orbImage = orbImageList[i];
            if(orbList.Count > i){
                orbImage.SetOrbFragments(1);
            }else{
                orbImage.SetOrbFragments(0);
            }
        }
    }
```

Listing 25: Functions for adding orbs to the orb list and refreshing the orb image list.

We also created a class for our orb sprites. Here, we defined the existence of two fragments, the orb sprite used to portray the fill of the score system and the other used to portray the decrease of the system (Listing 26).

```
public class OrbImage
    {
        private int fragments;
        private Image orbImage;
        private OrbBarCounterVisual orbBarCounterVisual;
        public OrbImage(OrbBarCounterVisual orbBarCounterVisual, 
Image orbImage)
        {
            this.orbBarCounterVisual = orbBarCounterVisual;
            this.orbImage = orbImage;
        }

        public int GetFragmentAmount() {
                return fragments;
            }
        public void SetOrbFragments(int fragments)
        {
            switch (fragments)
            {
                case 0: orbImage.sprite = 
orbBarCounterVisual.orbSpriteEmpty; break;
                case 1: orbImage.sprite = 
orbBarCounterVisual.orbSprite; break;
            }
        }
    }
```

Listing 26: Representation of one orb.

Once this was completed, we continued the process of developing the logic to complement the visual part.
Inside our Orb Bar System script, we had to define the max amount for our orbs, which was four. This means the player object can either lose or gather four orbs in total. Aditionally, we declared several events that defined the different states of the score system and we also created an orb list (Listing 27).

```
public class OrbBarSystem : MonoBehaviour {
    public const int MAX_FRAGMENT_AMOUNT = 4;
    public event EventHandler OnDamaged;
    public event EventHandler OnHealed;
    public event EventHandler OnRestart;
    public event EventHandler OnDead;
    private List<Orb> orbList;
```

Listing 27: Variables for our Orb Bar System script.

After this was done, we had to create a function named “OrbBarSystem()” that took as a parameter the “orbAmount” variable. This function was responsible for creating a new orb list and the adition of new orbs to the list (Listing 28).

```
public OrbBarSystem(int orbAmount)
    {
        orbList = new List<Orb>();
        for (int i = 0; i < orbAmount; i++)
        {
            Orb orb = new Orb();
            orbList.Add(orb);
        }
    }
```

Listing 28: Function that creates a new orb list and accepts orbs to be added to the list.

We then created a “Restart()” function, as well as a “Damage()”, “Dead()” and “Heal()” functions. These are important because we want to change the state of our Player object regarding the various events that can happen, such as catching an orb and healing due to it, or fighting against the Boss Enemy, losing all of the orbs and dying. The “Restart()” function is also very important because it allows us to clear the orb list when, for example, we unlock a puzzle or collide with a certain object (Listing 29).

```
public void Restart(){
         orbList.Clear();
         if (OnRestart != null) OnRestart(this, EventArgs.Empty);
    }
    public void Damage() {
        if(orbList.Count > 0){
           orbList.RemoveAt(orbList.Count-1);
        }
        if (OnDamaged != null) OnDamaged(this, EventArgs.Empty);
    }

    public void Dead() {
        if(orbList.Count < 0){
           StartCoroutine(DeadPlayer());
        }
    }

    public void Heal() {
        if(orbList.Count < MAX_FRAGMENT_AMOUNT){
            Orb orb = new Orb();
            orbList.Add(orb);
        }
        if (OnHealed != null) OnHealed(this, EventArgs.Empty);
    }
```

Listing 29: Functions for the various events that catching or losing an orb result in.

Once the logic for the score system was created, we had to apply it to our Player object, our orbs, enemies and the puzzles that only get unlocked if the player has a certain number of orbs. Inside our Player object, we called the "Heal()" and "Damage()" functions from the scripts we created previously in regards to the score system. Aditionally, and in regards to the "TakeDamage()" function, we also set the Player to play an animation of being hurt if they were damaged (Listing 30).

```
public static void Heal() {
      OrbBarCounterVisual.orbsHealthSystemStatic.Heal();
    }
public static void TakeDamage() {
      OrbBarCounterVisual.orbsHealthSystemStatic.Damage();
      animation.SetTrigger("hurt");
    }
```

Listing 30: Orb Bar System functions inside the PlayerPlatformerController script.

Inside our orb objects, we created a script that allows the Player object to be healed. We also created a visual effect to be triggered every time the player and the orb collided. After the collision, the orb dissapears and the score system increases by the number of orbs that were caught (Listing 31).

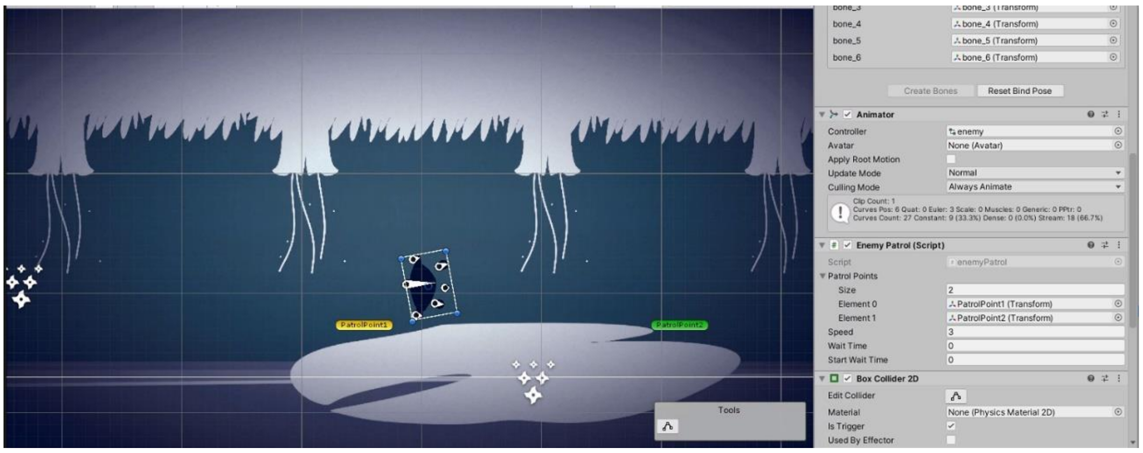
```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class Heal : MonoBehaviour
{
    //Visual Effect
    public GameObject orbEffect;

    private void OnTriggerEnter2D(Collider2D other) {
        Instantiate(orbEffect, transform.position,
Quaternion.identity);
        PlayerPlatformerController.Heal();
        Destroy(gameObject);
    }
}
```

Listing 31: Orb Bar System functions inside the PlayerPlatformerController script.

Inside our enemies (Fig.38), we stated that they would damage the player and, consequently, subtract orbs out of the score system (Listing 32).



```
bone_3          bone_3 (Transform)
bone_4          bone_4 (Transform)
bone_5          bone_5 (Transform)
bone_6          bone_6 (Transform)

Animator
Controller          enemy
Avatar              None (Avatar)
Apply Root Motion   ☐
Update Mode         Normal
Culling Mode        Always Animate
  Clip Count: 1
  Curves Pos: 6 Quat: 0 Euler: 3 Scale: 0 Muscles: 0 Generic: 0 PPtr: 0
  Curves Count: 27 Constant: 9 (33.3%) Dense: 0 (0.0%) Stream: 18 (66.7%)

Enemy Patrol (Script)
Script              enemyPatrol
Patrol Points
  Size              2
  Element 0         PatrolPoint1 (Transform)
  Element 1         PatrolPoint2 (Transform)
Speed               3
Wait Time           0
Start Wait Time     0

Box Collider 2D
Edit Collider
Material            None (Physics Material 2D)
Is Trigger          ✓
Used By Effector    ☐
```

Fig. 38: Patrol Enemy that deals damage to the main character and subtracts the orb list count.

```
private void OnTriggerEnter2D(Collider2D collision) {
    if (collision.gameObject.tag == "Player") {
        PlayerPlatformerController.TakeDamage();
    }
}
```

Listing 32: When the Player object collides with the Enemy object, the player takes damage.

Last but not least, we recall the script that stated that the player needed four orbs (the full score system) in order to open a certain puzzle. After that, the score system would be cleared out and the player would proceed into another level (Listing 33).

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class DoorScript : MonoBehaviour
{
    private Animator _animator;
    private bool hasBeenOpen = false;
    public GameObject noPassCollider;
    void Start() {
        _animator = GetComponentInChildren<Animator>();
        noPassCollider.SetActive(true);
    }
     private void OnTriggerEnter2D(Collider2D player) {
         if(!hasBeenOpen && player.gameObject.tag == "Player" &&
OrbBarCounterVisual.orbsHealthSystemStatic.getActiveOrbs() == 4) {
         noPassCollider.SetActive(false);
         hasBeenOpen = true;
         _animator.SetBool("open", true);
         OrbBarCounterVisual.orbsHealthSystemStatic.Restart();
         if(GetComponentInChildren<PolygonCollider2D>() != null)
         GetComponentInChildren<PolygonCollider2D>().enabled =
false;
         }
     }
}
```

Listing 33: Script attached to the game puzzle shows the Player needs four orbs to open the door.

With this work done, we managed to create multiple interactions of our main character with the rest of the game world, creating a more immersive and responsive experience.

4.2.9 Enemies

Enemies were a fulcral element to be added inside the game because they gave another purpose to the score system. Not only this, it made the game more dynamic in the sense that it gave more objectives to the player. Given this, we created a small patrol enemy, that guards a certain area of the game, and a final Boss, that serves as the final step towards the completion of the game. While the small patrol enemy serves the purpose of informing the player that they can, indeed, be damaged if they collide with any enemy, when the player encounters the second boss, the Boss Enemy, they already know what to do in order to run away from damage. For our patrol enemy, we had to determine two patrol points they would navigate through back and forward. If the Player object collided with this enemy, one orb would be lost from the score counter (Listing 34).

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class enemyPatrol : MonoBehaviour
{
   public Transform[] patrolPoints;
   public float speed;
   int currentPointIndex;
   public float waitTime;
   public float startWaitTime;

   private void Start() {
       transform.position = patrolPoints[0].position;
       waitTime = startWaitTime;
       transform.rotation = patrolPoints[0].rotation;
   }
   private void Update() {
       transform.position = Vector2.MoveTowards(transform.position,
patrolPoints[currentPointIndex].position, speed * Time.deltaTime);
       if (Vector2.Distance(transform.position,
patrolPoints[currentPointIndex].position) < 0.2f)
        {
            transform.rotation =
patrolPoints[currentPointIndex].rotation;
            if (waitTime <= 0) {
                if (currentPointIndex + 1 < patrolPoints.Length) {
                currentPointIndex++;
            }
            else {
                currentPointIndex = 0;
            }
            waitTime = startWaitTime;
            }
            else {
                waitTime -= Time.deltaTime;
            }
       }
   }

private void OnTriggerEnter2D(Collider2D collision) {
    if (collision.gameObject.tag == "Player") {
        PlayerPlatformerController.TakeDamage();
    }
}
}
```

Listing 34: Enemy Patrol script.

Likewise, the final Boss Enemy damages the Player object in units of one and can kill the Player if the orb count reaches zero (Listing 35), something that we made impossible for the collisions that happen with the normal small enemy.

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class BossScript : MonoBehaviour
{
      Rigidbody2D rb;
      private Animator enemyAnimations;
      public bool isFlipped = false;
      [SerializeField] float attackPlayerSpeed = 2;
      [SerializeField] float attackRange = 3;
      [SerializeField] Transform player;
      private Vector2 playerPosition;
      private int damage = 1;
      private Vector2 target;
      PlayerPlatformerController playerCont;

      void Start()
       {
            rb = GetComponent<Rigidbody2D>();
            enemyAnimations = GetComponent<Animator>();
            target = new Vector2(player.position.x,
transform.position.y);
            playerCont =
GetComponent<PlayerPlatformerController>();
       }

      public void AttackPlayer() {
            if (Vector2.Distance(player.position, rb.position) <=
attackRange && !isDead) {
                  StartCoroutine(DamageTime());
            }
      }
      IEnumerator DamageTime() {
            yield return new WaitForSeconds(1);
            enemyAnimations.SetTrigger("attack");
            PlayerPlatformerController.TakeDamage();
            yield return new WaitForSeconds(1);
      }
       private void Update() {
            AttackPlayer();
       }
}
```

Listing 35: Enemy Patrol script.

4.2.10 Monologue

Like we said in previous sections, the main character is alone in this misterious, dark world. Thus, they don't have anyone to talk to, nor ask where they are. Therefore, the character, Sirah, places all the questions she has as thoughts she shares through the use of text that we display. We developed this system using the canvas system of the Unity Engine and multiple colliders that make the text pop every time the Player object collides with them. In order to achieve this, we had to create a large array of colliders that we spreaded around the game world (Fig.39).

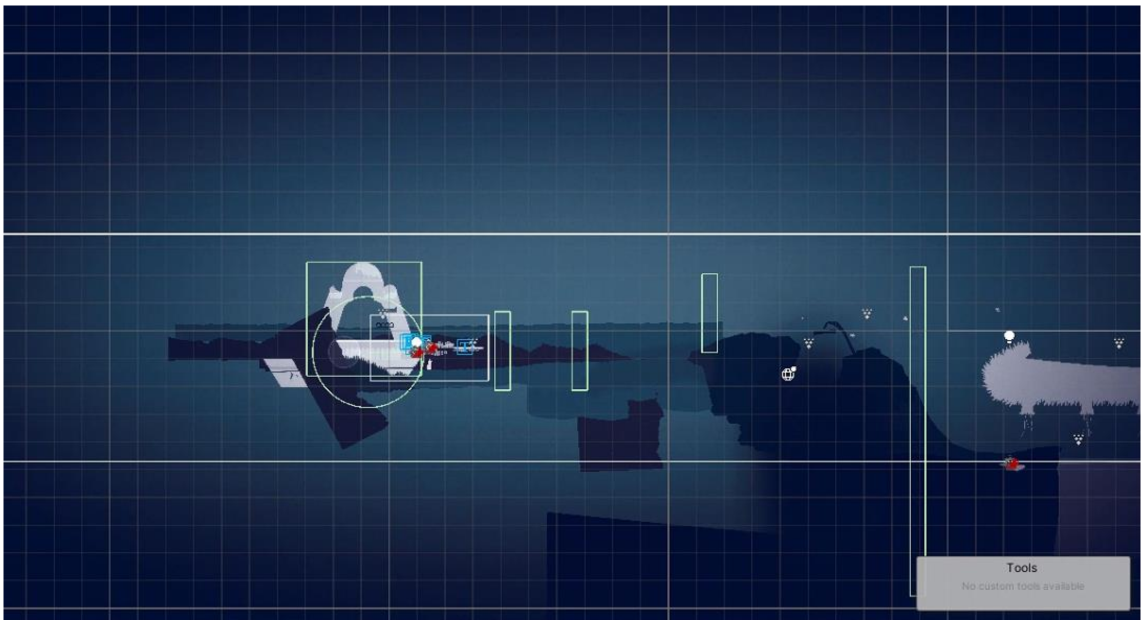


Fig. 39: Unity Scene View: Colliders spread around the game world.

Apart from this, we had to write some logic to make the text appear and disappear while the Player object was navigating around. In order to do this, we had to create a public game object named uiObject inside every single one of our colliders and selected the object we wanted to appear (in this case, the text). We then had to set this object as not being active because we didn't want it to appear at all times, only when the Player object entered certain colliders. After this, we confirmed if the object that was entering our text collisions was indeed our Player object and if this were true, then we would set the game object to be true, resulting in the text appearing and disappearing after three seconds (Listing 36).

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.UI;

public class texto : MonoBehaviour
{
    public GameObject uiObject;
    // Start is called before the first frame update
    void Start()
    {
        uiObject.SetActive(false);
    }
    // Update is called once per frame
    void OnTriggerEnter2D (Collider2D player)
    {
        if (player.gameObject.tag == "Player")
        {
            uiObject.SetActive(true);
            StartCoroutine("WaitForSec");
        }
    }
    IEnumerator WaitForSec()
    {
        yield return new WaitForSeconds(3);
        Destroy(uiObject);
        Destroy(gameObject);
    }
}
```

Listing 36: Enemy Patrol script.

This way, we created a smooth and responsive monologue system that tells the story of our main character, Sirah.

4.2.11 Boss Fight

We always wanted to create a boss that represented many things. First, we wanted the main character to defeat their Depression, portrayed in the shape of a big bird. Second, we wanted to show their relationship and make it so that the players could understand that the Enemy Bird Boss knew Sirah for a long time, and that it's been around her for a while. Third and last, it was an objective of ours to show that Depression, portrayed by the Bird, was there to kill Sirah, and was doing everything they could in order to do so. This said, we created a dialogue system between the Bird and Sirah, where the Bird, Depression, shares their objectives regarding Sirah. Sirah doesn't want to die and fights for her life. When she wins the battle, she wakes up and tells her story through the means

of text of how she woke up in an hospital bed, scarred, dressed in a white robe. This level we just described is the last one that occurs before the end of the game.
We started by importing our Bird sprite and created the bone structure and animations for it inside Unity. After this was done, we created numerous states that managed the animations and their states. Therefore, we ended up with five states: the initial idle animation, the run animation, the damage animation, the stomp animation and the enraged animation. We also created four parameters: "CutsceneIsActive", a boolean, "attack" a trigger, "damage", also a trigger, and "isEnraged", another boolean. The first animation to be played is the idle animation. When this animation is completed, the bird starts playing the run animation. After this, if the Boss is being damaged, then the animation of the Damage starts playing and when this finishes up, it returns to the run animation. Aditionally, the animations can go from run to enraged and vice versa, with the trigger being the boolean "isEnraged" being true. From here, the Boss can also start the stomp animation, triggered by the "attack" parameter (Fig.40).

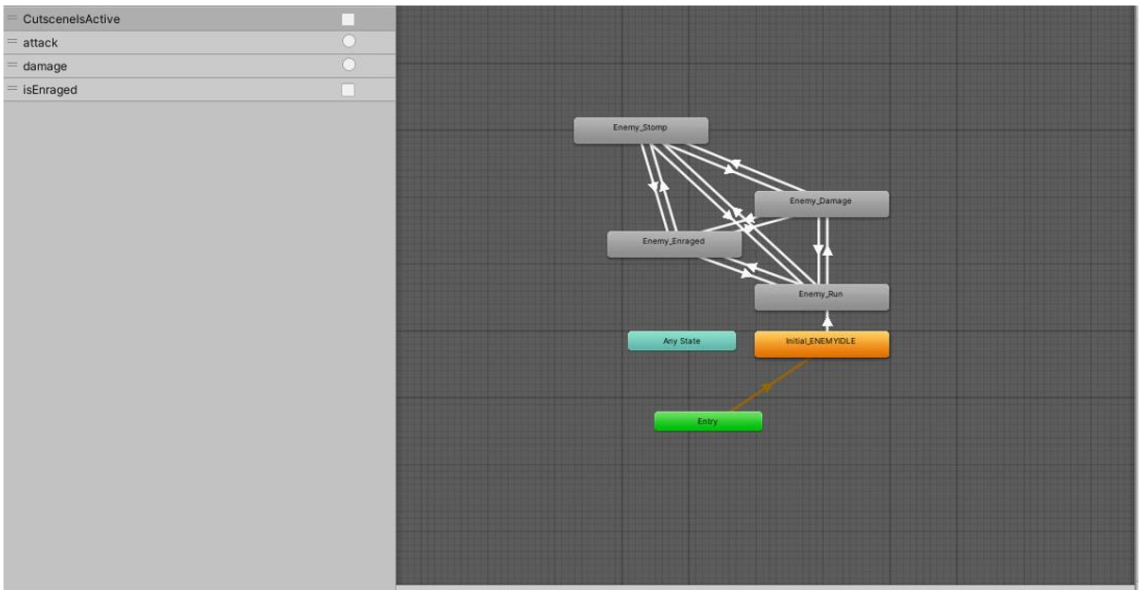
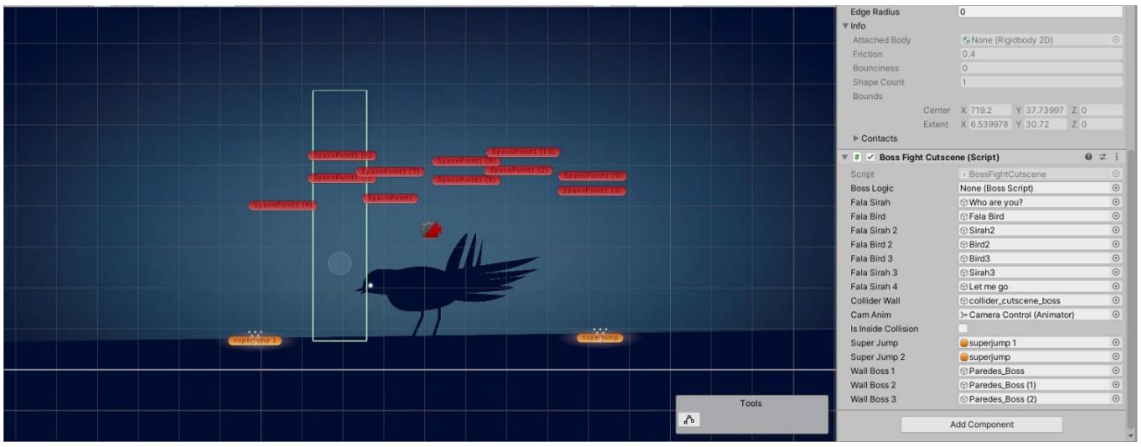


Fig. 40: Boss Enemy Animator.

With the animations and animator controller set up, we started developing the Boss Enemy script, as well as creating the colliders for the start of our Boss Fight Cutscene.
Regarding the cutscene that plays before the final fight, we created a collider that, when in contact with our Player object, would start this whole level. When this collision is entered, the Bird starts speaking with Sirah. The Bird makes their point clear and Sirah has to run away from them. At this point, we change cameras for a wider one, so that the player can see the level better and reach the platforms that are high in the sky (Fig.41).



Edge Radius	0
▼ Info	
Attached Body	None (Rigidbody 2D)
Friction	0.4
Bounciness	0
Shape Count	1
Bounds	
Center	X 719.2 Y 37.73997 Z 0
Extent	X 6.539978 Y 30.72 Z 0
▶ Contacts	
▼ ✓ **Boss Fight Cutscene (Script)**	
Script	BossFightCutscene
Boss Logic	None (Boss Script)
Fala Sirah	Who are you?
Fala Bird	Fala Bird
Fala Sirah 2	Sirah2
Fala Bird 2	Bird2
Fala Bird 3	Bird3
Fala Sirah 3	Sirah3
Fala Sirah 4	Let me go
Collider Wall	collider_cutscene_boss
Cam Anim	Camera Control (Animator)
Is Inside Collision	☐
Super Jump	superjump 1
Super Jump 2	superjump
Wall Boss 1	Paredes_Boss
Wall Boss 2	Paredes_Boss (1)
Wall Boss 3	Paredes_Boss (2)

Add Component

Fig. 41: Boss Enemy Final Fight scene view.

After this cutscene collision is detected, two light spots that increase the jump force appear. When the Player object enters these, the player can jump higher and get to the top of the platforms that are suspended in the sky. Moreover, when a collision is detected between the Player object and the platform collider, the platform starts falling, hits the Bird and causes damage to them. The Bird has 20 life points. This means the player has to hit the enemy with 20 falling platforms. When the enemy's life reaches its' end, the final Bird animation and dialogue is unlocked. In this portion, the Bird says goodbye to Sirah and explodes into a mist of orbs (see Attachment 1).

The platforms the Player object can jump into and make them fall on top of the Bird, damaging them, were created in order to respawn every time the number of platorms was zero, this is, every time there wasn't a platform available to be jumped on. In order to do this, we had to create several spawn points (presented as red gizmos in the image below) and created a script that determined they should appear in a random order. This made the game a bit harder as we intended, because the player cannot find patterns between the platforms (Fig.42).

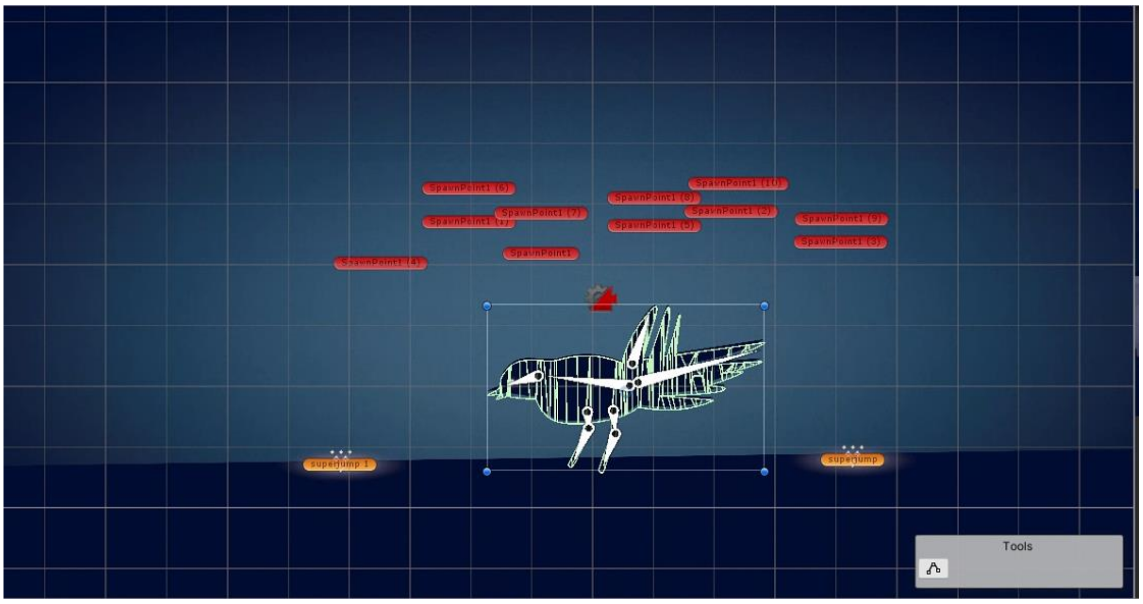


Fig. 42: The red gizmos represent the points where the platforms will spawn.

If the Player object gets too close to the Bird Enemy, the last one will start damaging the player. If the Player object reaches a life count of zero, the player dies and has to restart the level. In order to prevent the player from dying, we added a set of orbs that respawns when they are all consumed in the middle of the level view. This way, the player can counter the Boss with the help of some healing.

4.2.12 Checkpoints

Creating checkpoints is important because we don't want to obligate the player to play the game repeatedly if, for any instance, they make the character die by falling off a platform or taking too much damage from the Boss Enemy. With this in mind, we created several colliders placed in strategic places around the game world. When the Player object enters this collision, the position of the Player object was last in is detected and the Player respawns in this position if they fall and die (Listing 38).

```
IEnumerator RespawnPlayer() {
        Instantiate(playerDestroyed, transform.position, 
Quaternion.identity);
        yield return new WaitForSeconds(1);
    }

private void OnTriggerEnter2D(Collider2D other) {
      if (other.gameObject.tag == "CheckPoint") {
            respawnPosition = other.transform.position;
      }
      if (other.gameObject.tag == "CatchZone") {
            isDead = true;
            StartCoroutine(RespawnPlayer());
            transform.position = respawnPosition;
        }
    }
```

Listing 38: Player Platformer Controller Script: Respawn the Player if they die.

This way, the player can respawn in the closest point from where the character died, making the game less tiresome.

4.2.13 Menus

Menus are an essential asset when creating a game as a whole. With access to the menus that we created, the player can start the game, pause it and leave it any time they desire. This gives the player the power to chose when they want to play and how they want to play. We created both an initial menu as well as a menu that can be accessed any given time by clicking the "ESC" key. As a means to create these menus, we had to create two other scenes prior to the ones we had already created for our two levels and cutscene. Inside the scene for our Main Menu (the first menu the player can interact with), we created a UI Canvas Game Object. Then, inside this game object we created an array of UI Image game objects. These images were used in order to build the background of the game menu. After this, we created two more UI Image game objects, that we used in order to create the "Start Game" button and the "Quit" button. Since we want these two images to be buttons, we have to add the "Button" component to them. When we completed this task, we added a script to our parent Canvas game object and named it "Menu". Inside this script we created a function that would load the various scenes we have in our game ("LoadScene()") and another one that would just quit the game application ("Quit()"). Furthermore, we created a coroutine that handled the transition between this scene and the one after it (the first level of the game) (Listing 39).

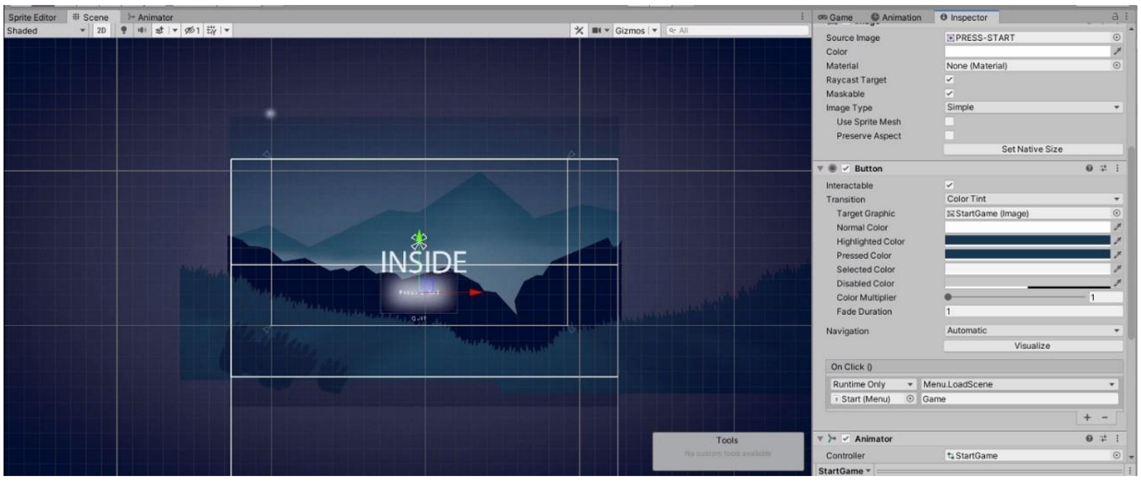
```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.SceneManagement;

public class Menu : MonoBehaviour
{
    public GameObject panel;
   public void LoadScene(string sceneName) {
       StartCoroutine(fadein(sceneName));
   }
   public void Quit() {
       Application.Quit();
   }

   IEnumerator fadein (string sceneName) {
    panel.SetActive(true);
    yield return new WaitForSeconds(3f);
    SceneManager.LoadScene(sceneName);
   }
}
```

Listing 39: Script for the Menu.

When this step was finalized, we had to open our File Build Settings inside Unity and selected all of the scenes that would be part of the final build. With this done, we added an “On Click” event that is attached to the Button component placed in both of our button images. We placed the canvas game object inside the “On Click” event slot and selected the “LoadScene” function as the function that would be triggered. After this, we had to assign a scene to be loaded. In the case of the “Start Game” image we assigned the “Game” scene to it, because this is the scene that has our first level (Fig.43).



Source Image: PRESS-START
Color
Material: None (Material)
Raycast Target: ✓
Maskable: ✓
Image Type: Simple
Use Sprite Mesh
Preserve Aspect
Set Native Size
Button ✓
Interactable: ✓
Transition: Color Tint
Target Graphic: StartGame (Image)
Normal Color
Highlighted Color
Pressed Color
Selected Color
Disabled Color
Color Multiplier: 1
Fade Duration: 1
Navigation: Automatic
Visualize
On Click ()
Runtime Only | Menu.LoadScene
Start (Menu) | Game
Animator ✓
Controller: StartGame
StartGame

Fig. 43: Unity Scene View: Main Menu.

The menu triggered on pause was created according to the same standards. The only difference is that this game menu could be accessed when the “ESC” key was triggered and there are two navigation options: the player resumes the game or quits it.

4.2.14 Transitions

Transitions helped us create a smoother game that looks well together as a whole composition. Without them, we would have only bits and pieces that preceded and proceded various stages. In our game, transitions were created with the help of a UI Panel game object, which is basically just a big overlay. We set the color to be black in some instances, such as the transition between the game menus and the levels, while in others we made it white, as we can see in the completion of the first level going forward the second. Furthermore, some transitions go from a full color to a transparent gradient while others go from this last on all the way into a full color. With the help of the “Animation” window we created these variations changing the opacity of the panel. The default state of the transitions is not active, so we can activate them whenever we want. This way, we implemented this effect in any scene we desired.

4.2.15 Visual Effects

Visual effects add beauty and life to the game. The visual effects present in our game were generated with the help of Unity’s Particle Effects. In our game, we created a visual effect for the collisions between the Player object and the orbs, as well as the first one and the respawn collisions. Aditionally, we also created a particle system that would be triggered every time a platform was destroyed or collided with another object.

4.2.16 Post Processing

Post Processing helped us give a moodier and more beautiful look to the game. The grain, the saturation of the colors and the vignette gave the game a nostalgic look and the darkness of the whole screen portrays the darkness of the subject that is being developed about. Post Processing allows the difference between a game that does not catch the eye (Fig.44) into one that does (Fig.45).

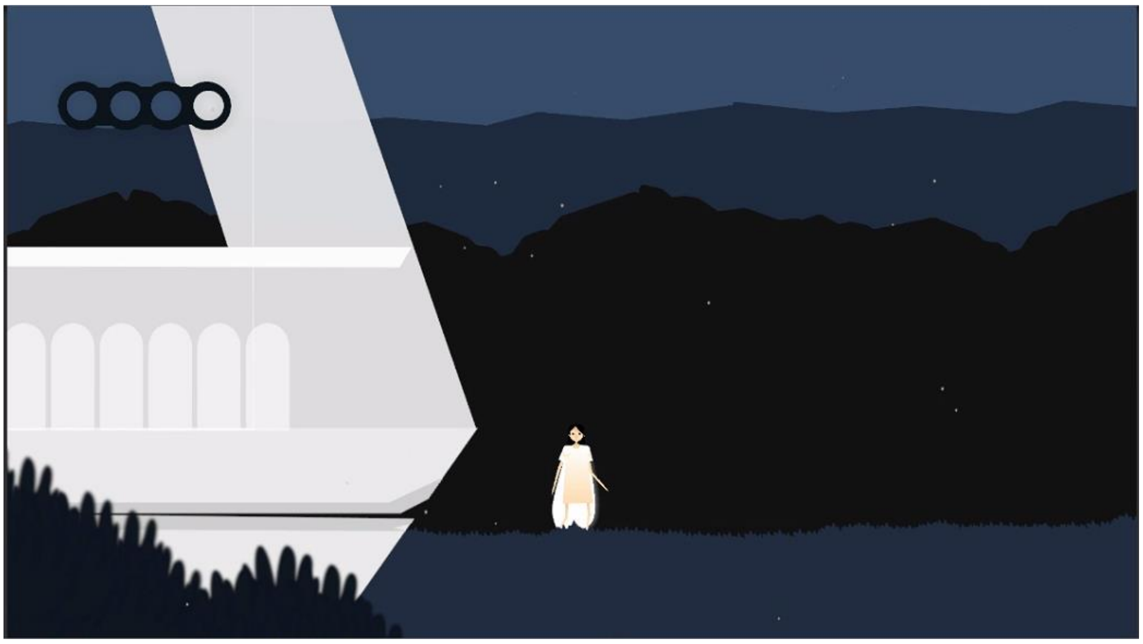


Fig. 44: Game View without Post Processing.

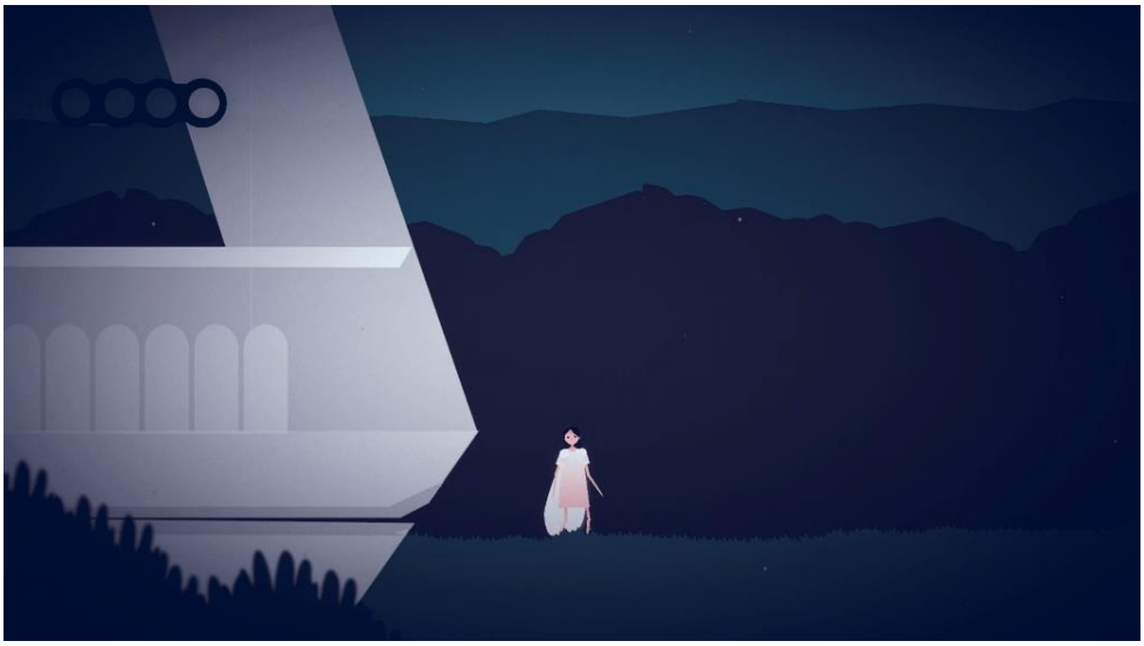


Fig. 45: Game View with Post Processing.

4.2.17 Music and Sounds

Our objective for the soundtrack of the game was always creating a personal and personalized audio composition made by us. Therefore, we created all of the music that can be heard inside the video game. In order to build a mood and environment that combined with, and, most importantly, enhanced the visual part of the game, making it cohesive as a whole, we had to choose between some mellow instruments played in lower tones, such as the piano. We created a melancholic sound for our game because it seemed to fit the theme being handled. The other sounds that can be heard were taken out of ambient sounds, such as the wind or the rain. The music was created using a MIDI keyboard and the software Soundtrap. We added a lot of distortions to the sound of the piano, in order for it to sound deeper. The orb sound effect was created with the MIDI keyboard as well, and it consists of three simple key strokes. Apart from this, the sound of wind that can be heard in the background was downloaded from a free source.

Chapter 5

Conclusions and Future Work

With the game concluded, we could indeed assert that this project served as a tool for expression. The game, in some sorts, is autobiographical, hence the need to say the development was a matter of expression. We believe video games will increasingly be developed as a way of portraying the life of the people that develop them, just like we saw previously in the Related Work section. Artists have been sharing their realities and life for centuries through the means of musical instruments, paint and other tools. Just like other kinds of art, we believe video games can certainly represent a tool for storytelling, the voicing of problems and issues, and the exposition of different themes. Overall, we believe this game operates as a personal project that portrays a story and a point of view and that this is the main point of art and using art as an expression tool. Regarding our educational objectives, we did not have time to access the educational power of the game in a test environment. Even though we tested the game with some friends, who declared they did indeed understand the purpose of the game and its meaning, we did not have time to test the educational purpose of the game within a contained and experimental environment. Therefore, this is a task we had to leave for a future time, when we have an extended period of time to test this hypothesis.

Lastly, this project showcased the abilities that were developed during the Game Design and Development Master’s Degree. With the knowledge that was gathered during the participation in the various courses of the Master’s Degree, we found it possible to develop a full game independently.

Bibliography

Aarseth, E. (1997). Cybertext: perspective on ergodic literature [Electronic Version]. *Johns Hopkins University Press*, UK ed. Edition. Accessed in 17 of May of 2019 at: https://pt.scribd.com/document/266117002/Espen-J-Aarseth-Cybertext-Perspectives-on-Ergodic-Literature

Adobe Photoshop (2020). Accessed in 14 of September of 2020 at: https://www.adobe.com/pt/products/photoshop.html

Adobe Premiere (2020). Accessed in 14 of September of 2020 at: https://www.adobe.com/pt/products/premiere.html

Akins, W., Fadhlurrahman A. and Clifton, N. (2018). *Get Help*. Accessed in 14 of September of 2020 at: https://wakins.itch.io/get-help

Blumberg, F.C. (2014). Learning by playing: video gaming in education [Electronic Version]. *Oxford University Press*, Illustrated Edition (April 7, 2014). Accessed in 16 of November of 2019 at:
https://books.google.pt/books?hl=pt-PT&lr=&id=ptnQAgAAQBAJ&oi=fnd&pg=PP1&dq=educational+video+games&ots=Sa81_cideC&sig=ytERUGsAtiUlKJ1Wyw5RYCNd56U&redir_esc=y#v=onepage&q=educational%20video%20games&f=false

Christiansen, P. (2017). *Designing ethical systems for videogames*. Accessed in 3 of May of 2019 at:
https://www.academia.edu/36321095/Designing_Ethical_Systems_for_Videogames

Crawford, C. (2000). The Art of Computer Game Design. Accessed in 17 of November of 2019 at:
http://www.stonetronix.com/gamedesign/art_of_computer_game_design.pdf

Creations, F. (2018). *To Leave*. Accessed in 14 of September of 2020 at: https://store.steampowered.com/app/896340/To_Leave/

Dragon Bones (2017). Accessed in 14 of September of 2020 at: http://dragonbones.com/en/index.html

Egret Developer (2017). Dragon Bones API. Accessed in 14 of September of 2020 at: https://developer.egret.com/en/apidoc/index/name/dragonBones.EventObject.FADE_IN

Eichenbaum, A., Bavelier, D., Green, C.S. (2014). *Video games: play that can do serious good*. Accessed in 16 of November of 2019 at: https://archive-ouverte.unige.ch/unige:84313

Freeland, C. (2001). But is it art? An Introduction to Art Theory. Acessed in 11 of November of 2020 at: https://kupdf.net/download/cynthia-freeland-but-is-it-art-pdf_59f2168fe2b6f555715122f0_pdf

GitHub (2016). Dragon Bones Plugin. Accessed in 09/06/2020 at: https://github.com/DragonBones/DragonBonesUnity

Global Health Metrics (2018). *Global, regional, and national incidence, prevalence, and years lived with disability for 354 diseases and injuries for 195 countries and territories, 1990-2017: a systematic analysis for the Global Burden of Disease Study 2017*. Accessed in 14 of September of 2020 at: https://www.thelancet.com/journals/lancet/article/PIIS0140-6736(18)32279-7/fulltext

Grimes, A. (2014). *What Now?*. Accessed in 14 of September of 2020 at: https://ariellegrimes.itch.io/what-now

Heron, M.J. and Belford, P.H. (2014). *Do you feel like a hero yet? externalized morality in video games*. Accessed in 3 of May of 2019 at: https://static1.squarespace.com/static/51f9aac5e4b080ed4b441ba7/t/53b02ae2e4b0182daaf7f3c4/1404054242059/HeronBelford-1-2.pdf

Lonien, J. (2014). *I'm better*. Accessed in 14 of September of 2020 at: https://ludonaut.itch.io/i-am-better

Mattos, S. (2016). *Como elaborar objetivos de pesquisa*. Accessed in 16 of November of 2019 at:

http://unesav.com.br/ckfinder/userfiles/files/Como%20elaborar%20Objetivos%20de%20Pesquisa.pdf

Quinn Z., Lindsey, P. and Schankler I. (2013). *Depression Quest*. Accessed in 14 of September of 2020 at:
http://www.depressionquest.com/

Rauch, P. (2007). *Severe pain or suffering: videogames, morality and torture*. Accessed in 16 of May of 2019 at:
http://gamephilosophy.org/wp-content/uploads/confmanuscripts/pcg2007/Rauch_PaperPCG2007.pdf

Robot, G.E. (2017). *Distracting Myself From Suicide*. Accessed in 14 of September of 2020 at: https://giantevilrobot.itch.io/distracting-myself-from-suicide

Rokashi (2016). *I'm Fine*. Accessed in 14 of September of 2020 at: https://rokashi.itch.io/im-fine

Sabiarts (2017). *Kiss Me*. Accessed in 14 of September of 2020 at: https://sabiarts.itch.io/kissme-lovejam2017

Smuts, A. (2005). *Are video games art?* Accessed in 17 of November of 2019 at:
https://digitalcommons.risd.edu/cgi/viewcontent.cgi?article=1043&context=liberalarts_contempaesthetics

Soundtrap (2020). Accessed in 14 of September of 2020 at: https://www.soundtrap.com/

Squire, K. (2003). *Video games in education*. Accessed in 3 of May of 2019 at: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.543.5729&rep=rep1&type=pdf

TEGD. (2014). *Fallout 3*. Accessed in 3 of May of 2019 at: https://tegd.arizona.edu/index.php?title=Fallout_3

Unity (2020). Accessed in 14 of September of 2020 at: https://unity.com/

Wolf, M.J.P. (2017). *Video games as american popular culture*. Accessed in 03 of May of 2019 at:
https://rua.ua.es/dspace/bitstream/10045/68127/1/Quaderns-de-Cine_12_10.pdf

WZOGI (2014). Actual Sunlight. Accessed in 14 of August 2020 at:
https://willoneill.com/actualsunlight/

Zagal, J.P. (2009). *Ethically notable videogames: moral dilemmas and gameplay*. Accessed in 17 of May of 2019 at:
https://www.academia.edu/23100235/Ethically_Notable_Videogames_Moral_Dilemmas_and_Gameplay

Attachments

Attachment 1 - Listing 37: Boss Enemy Script

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class BossScript : MonoBehaviour
{
      BossFightCutscene bossCutscene;

      [Header("Other")]
      Rigidbody2D rb;
      private Animator enemyAnimations;
      public bool isFlipped = false;

      [Header("AttackPlayer")]
      [SerializeField] float attackPlayerSpeed = 2;
      [SerializeField] float attackRange = 3;
      [SerializeField] Transform player;
      private Vector2 playerPosition;
      private int damage = 1;
      private Vector2 target;

      [Header("EnemyLife")]
      public int health = 20;
      public bool isDead;
      public GameObject deadEffect;

      [Header ("Player")]
      PlayerPlatformerController playerCont;
      public GameObject finalEffect;

      public GameObject birdSpeak1;
      public GameObject birdSpeak2;
      public GameObject birdSpeak3;

      //public GameObject finalCamera;
      void Start()
    {
            rb = GetComponent<Rigidbody2D>();
            enemyAnimations = GetComponent<Animator>();
            target = new Vector2(player.position.x,
transform.position.y);
            playerCont =
GetComponent<PlayerPlatformerController>();
            //finalCamera.SetActive(false);
            bossCutscene = GetComponent<BossFightCutscene>();
            birdSpeak1.SetActive(false);
            birdSpeak2.SetActive(false);
            birdSpeak3.SetActive(false);
      }
```

Listing 37: Boss Enemy Script.

```
public void TakeDamageEnemy(int damage){
            if(health <= 20 ) {
                  health -= damage;
                  enemyAnimations.SetTrigger("damage");
            }
            if (health <= 0) {
                  DefeatedEnemy();
            }
      }
      public void DefeatedEnemy() {
            StartCoroutine(Dead());
            bossCutscene.StopCutscene();
      }
      IEnumerator Dead() {
            isDead = true;
            enemyAnimations.SetBool("isEnraged", true);
            yield return new WaitForSeconds(2);
            birdSpeak1.SetActive(true);
            yield return new WaitForSeconds(2);
            Destroy(birdSpeak1);
            yield return new WaitForSeconds(2);
            birdSpeak2.SetActive(true);
            yield return new WaitForSeconds(2);
            Destroy(birdSpeak2);
            yield return new WaitForSeconds(2);
            birdSpeak3.SetActive(true);
            yield return new WaitForSeconds(2);
            Destroy(birdSpeak3);
            yield return new WaitForSeconds(2);
            Instantiate(deadEffect, transform.position,
Quaternion.identity);
            Destroy(gameObject);
      }

      public void AttackPlayer() {
            if (Vector2.Distance(player.position, rb.position) <=
attackRange && !isDead) {
                  StartCoroutine(DamageTime());
            }
      }
      IEnumerator DamageTime() {
            yield return new WaitForSeconds(1);
            enemyAnimations.SetTrigger("attack");
            PlayerPlatformerController.TakeDamage();
            yield return new WaitForSeconds(1);
      }
```

Listing 37: Boss Enemy Script.

```
void FlipTowardsPlayer() {
            {
            Vector3 flipped = transform.localScale;
            flipped.z *= -1f;

            if (transform.position.x > player.position.x &&
isFlipped)
            {
                  transform.localScale = flipped;
                  transform.Rotate(0f, 180f, 0f);
                  isFlipped = false;
            }
            else if (transform.position.x < player.position.x &&
!isFlipped)
            {
                  transform.localScale = flipped;
                  transform.Rotate(0f, 180f, 0f);
                  isFlipped = true;
            }

      }
      }

       private void Update() {
            AttackPlayer();
            FlipTowardsPlayer();
       }
}
```

Listing 37: Boss Enemy Script.

Attachment 2 - Game Design Document for Inside – A Video Game about Depression

Inside
Game Design Document
First Version

Beatriz Botelho

Game Design Document according to Bob Gates in Game Design - Second Edition (2004)

Abstract: In this version of the Game Design Document, we find information regarding the First Version of the game that is being developed. The mechanics, levels, controllers, abilities, animations, visual effects and music contemplate the final version of the game, but there may be some alterations since the game is still being worked on.

1. Introduction

This document specifies the design for the gameplay of a game with the title "Inside". Inside is a game that portrays the life journey of a person who suffers from depression. The game illustrates a transversal problem that affects an increasing number of people. According to the World Health Organization, in 2017, 300M of people suffered from depression. To the author, this game and its development are important because depression is still taken as a minor illness, lacking importance for some. Depression is a serious illness that takes many lives early, so, speaking about topics like this is important.

2. Scope

This document is intended to be read by programmers, artists, game designers and producers involved in the design, implementation and testing of Inside.

3. SECTION I: PROJECT OVERVIEW

a. Team Personnel

i. Beatriz Botelho - Programmer, UI/UX, Artist, Animator, Game Design and Music Composer

- bbia20@outlook.pt

b. Executive Summary

i. High Concept

- A 2D Platformer built on a small world where the player follows Sirah, a girl who just commited suicide. The game portrays Sirah's journey through all the moments that led her to this situation.

In order to progress through the game, Sirah must collect enough Soul Energy Points in order to be able to unlock puzzles. Soul Energy Points can be collected either by completing the platforms, either by defeating enemies.

ii. The Hook

- A game that tells the story of Sirah, a comatose patient who commited suicide. The player follows Sirah through her journey to find help and heal from her illness, through small puzzles, well-thought mechanics and art.

iii. Story Synopsis and Setting

- Sirah suffers from depression and tries to commit suicide. In order to do so, she throws herself from a tall building. When she does this, she gains a pair of wings and enters the realm of the comatose. With the help of her new wings, she will be able to fly around her deep and most setted memories, and try to find a cure to her own disease. When Sirah lives through all of her memories and fights all of the small parts of the disease that got to manifest into her life, as she heals progressively, she is challenged by a final Enemy, the most deep rooted representation of her depression, and the hardest phase to defeat. When the player defeats the final and biggest enemy, Sirah wakes up in a hospital bed, touches her back in order to find the wings she had in the comatose dimension and finds some suture stitches.

iv. Genre and Scope (such as number of missions or levels)

- 2D puzzle platformer with 3 different levels, interconnected between areas that unlock after each area is completed, where each level represents a mission where the player needs to use the abilities available in order to complete it. There is also a final fight the player must face, where the player must use every ability he/she wishes in order to defeat Depression.

v. **Visual Style**

- The game will present itself in 2D sprites inspired by Gris and Rayman Legends.

vi. **Engine and editor**

- The game is being made in Unity.

c. ***Core Gameplay (What does the player do?)***

i.Single Player

- Solve puzzles.
- Collect Soul Points in order to unlock puzzles and abilities.
- Use different abilities to travel through a dark dimension making use of platforming sections to complete different levels..

d. ***Game Features***

i.Gameplay

- A 2D platform-adventure game that portrays the journey of Sirah, who suffers from depression and tries to defeat her illness during her coma. The player can explore the world at his/her own pace and engage in activity whenever he/she desires. The final battle against depression is only possible after the player has gone through the entire game and completed all the puzzles that are present in the game.

ii. **Artistic techniques and achievements**

- 2D simple sprites, stylized with moody, dark shadows and mellow/melancholic colors.

iii. **Other features that will make this game better than others like it on the market**

- The game story. Sirah's journey is meant to represent a catharsis and healing from depression.

e. ***Project Scope***

i.Number of distinct locations

- Three interconnected locations where the player can collect Soul Points, unlock find different abilities and use them the way he/she wishes in order to fulfill certain objectives.

ii. **Number of levels/missions**

- Three levels. The final level is the final battle against Depression, where Sirah will confront her illness.

iii. **Number of NPCs**

- World NPCs: Creatures from the domain: Birds, fireflies.

iv. **Number of abilities**

- 5 different abilities that help the player progress through the game: Jump, Run, Wing Jump, Attack, Power Flight.

v. **Number of songs and sounds**

- Custom soundtrack : To be defined..
- **Sound Effects:**
  Sirah: Jump, Run, Wing Jump, Attack, Power Flight.
  Environmental sounds: Wind Sound, Trees Leaves moving with the wind.

f. Target Audience

- People in an age range between 15 and 50 who have ever suffered from Depression or any other type of mental illness.

g. Delivery Platform(s)

- Windows PC and Mac OS

4. SECTION II: STORY, SETTING, AND CHARACTER

a. Story

i. Back story

- Sirah has lived her entire life with depression. She has endured the illness for most of her life but she gave in and tried to commit suicide.
  - **In-game story (What happens during the game)**
    - When Sirah tries to commit suicide, she gains a pair of wings and enters the realm of the comatose. With the help of her new wings, she will be able to fly around her deep and most setted memories, and try to find a cure to her own disease. Sirah lives through all of her memories and fights all of the small parts of the disease that got to manifest into her life, as she heals

progressively, she is challenged by a final Enemy, the most deep rooted representation of her depression, and the hardest phase to defeat. When the player defeats the final and biggest enemy, Sirah wakes up in a hospital bed, touches her back in order to find the wings she had in the comatose dimension and finds some suture stitches.

- ***Environments (Order subject to change)***
  - **Location 1 - Memory Forest**
    - **General description:**
  - A dark forest filled with falling trees and tricky platforms that are about to fall. Here, the player can gather Soul Points and learn how to move Sirah, as well as jump and double jump (Wing Jump).
    - **Physical characteristics:**
  - A dark, moody place filled with large mountains and trees. In this area, Sirah learns how to use her Wing Jump by collecting Soul Points.
    - **List of levels that take place in this area:**
  - Level 1
  - **Location 2 - Deep Waters**
    - **General description:**
  - Sirah falls from a creek and finds this place. In this area, the player can find water sheets and large peace lilies that can be used as platforms. In this area, Sirah must unveil a puzzle in order to go to the next area.
    - **Physical characteristics:**
  - An underground magical place filled with flowers and light.
    - **List of levels that take place in this area:**

- Level 2

- **Location 3 – Boss Fight**
    - **General description:**

- A large podium where Sirah and Depression fight till death.

    - **Physical characteristics:**

- Dark, melancholic and large. There are platforms from which Sirah can jump in order to hurt Depression.

    - **List of levels that take place in this area**

- Final Boss Fight

- ***Characters***
    - **Playable Character**
- Sirah
    - **Personality**
- Quiet, brave.
    - **Back story**
- Sirah suffers from depression and tries to commit suicide. In order to do so, she throws herself from a tall building. When she does this, she gains a pair of wings and enters the realm of the comatose.

    - **"Look"**
- Has her hospital garment on. Has a floaty feeling to her movement. Is surrounded by a bright light.
    - **Special abilities**
        - ***Ability #1 - Wing Jump (Double Jump)***
            - **When it's acquired:**

Sirah acquires this ability during the first level.

- **How the player invokes it:**
- Pressing the space bar after a jump has been invoked.
    - **Effect it has on the world**
- Makes Sirah dash vertically in the form of a double jump. This ability can be used in order to reach different (higher) levels she couldn't before.

    - **Graphics effect that accompanies it**
- Wind summons under Sirah's garment. This graphic effect is accompanied by double jump sound.

- ***Ability #2 - Power Flight***
    - **When it's acquired:**

  Oria acquires this ability during the second level
    - **How the player invokes it:**
- This ability can be invoked by pressing the space bar for a long time and then releasing it.
    - **Effect it has on the world**
- The jump boost can be seen in form of dust and wind coming from Sirah's legs as she prepares to fly.
    - **Graphics effect that accompanies it**
- Wind and Dust. Boost and wind sound can be heard.

- ***Ability #3 - Light Ability***
    - **When it's acquired:**

  Sirah acquires this ability during the second level
    - **How the player invokes it:**
- This ability can be invoked by pressing E.
    - **Effect it has on the world**
- Sirah gets filled with Light and destroys all the darkness that may be present in a certain area.
    - **Graphics effect that accompanies it**
- Bright White Light.
- **Regular Animations**
    - Walk, run, crouch, idle, jump.
- **Situation-specific animations**
- Sirah uses Wing Jump
- Sirah uses Power Flight

- **Enemies**
    - Depression
        - Personality:
    - Quiet, tactical.
        - Relationship to player character:
    - Sirah's illness.
        - Back Story:
    - The illness has affected Sirah for years and has led her to commit suicide.
        - "Look":
    - At first, a white, bright bird that transforms into a dark, ravenous bird, leaving dark goo everywhere it goes.
        - Special abilities:

- Powerful Scream, Floor Stomp (making it harder for Sirah to climb the platforms), Fly Attack, Goo Floor.
  - Regular animations:
- Idle, Heavy Breathing
  - Situation-specific animations:
- Taking Damage
- Power Flight
- Scream
- Floor Stomp

- **Neutrals**
  - World NPCs: Creatures from the domain: Birds, fireflies.

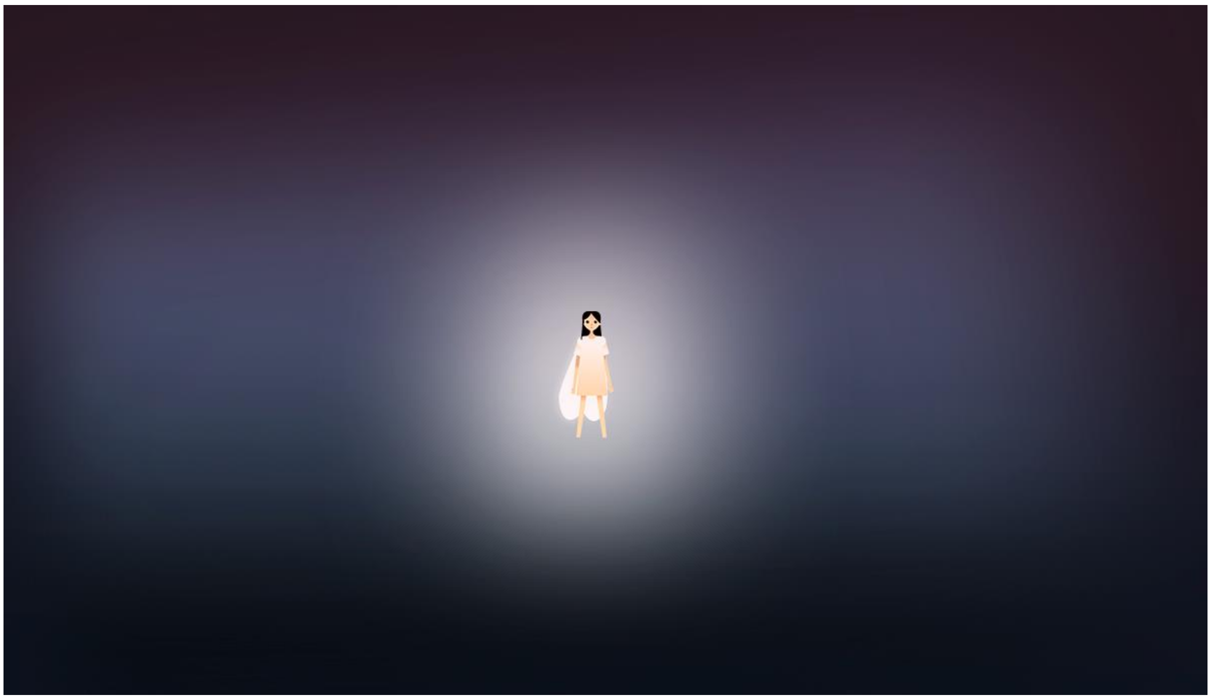
5. SECTION III: COMBAT

b. Abilities

i.Light Ability

1. General description and most effective use:
   - This ability can be used in order to fight against Depression. When the player invokes this ability, Sirah gets filled with Light and destroys all the darkness that may be present in a certain area.
2. When it is first acquired
   - Sirah acquires this ability during the second level.

3. Art:



ii. **Power Flight**

1. General description and most effective use:
   - This ability can be invoked by pressing the space bar for a long time and then releasing it. Sirah jumps with strength and flies for a certain period of time.
2. When it is first acquired
   - Level 2
3. Art:

iii. **Wing Jump**

1. General description and most effective use:
   - This ability can be invoked by pressing the space bar quickly after the first jump has been invoked. Sirah can now double jump.
2. When it is first acquired:
   - Sirah acquires this ability in the first level.
3. Art:

6. SECTION IV: CONTROLS

c. *PC Keyboard/Mouse Commands*

i.Default keys for movement controls

1. Move forward: W
2. Move Backward: S
3. Turn Left: A
4. Turn Right: D
5. Jump: Space
6. Crouch: F

ii. Default keys for using abilities

1. Wing Jump: Doube Press Space Bar
2. Power Flight: Continuously press Space Bar for an extended period of time
3. Light Ability: E

iii. Menu access

- Esc

7. SECTION V: INTERFACE

d. *The Camera*

i.Standard view:

- Third Person

ii. Player-controllable options

- Main Menu
- Pause Menu

e. *HUD*

i.Worldview (what the player sees)

- To be defined. At the moment, the game developer does not intend on having any visual queues for these.

ii. Status information

- To be defined. At the moment, the game developer does not intend on having any visual queues for these.

f. *Menus*

i. Game screen flow diagrams

ii. Start Menu

1. Start Game button
2. Exit button

iii. Pause Menu

1. Resume
2. Quit

iv. Credits

8. SECTION VI: ARTIFICIAL INTELLIGENCE (AI)

g. *Dragon Boss*

i. Statistics

1. Field of view: 360
2. Range of View: To be Defined

ii. Internal states & the triggers that change them

1. Idle: State between attacks
2. Scream: Depression screams at the player every time he/she gets to close to it.
3. Floor Stomp: Invoked every time the player manages to get close to Depression (enemy battle) with the use of platforms present in the area.
4. Power Flight: Invoked in order to deal damage to Sirah and disorientate her.

iii. Combat decisions

1. Foe recognition: Follows player position before and after attacking.
2. Targeting decisions: Scream and Floor Stomp is targeted at the player.
3. Attack: Power Flight has a cooldown before it can be used again by Depression.

9. SECTION VII: DETAILED LEVEL/MISSION DESCRIPTIONS

h. *Level #1*

i. Synopsis

- The game is divided in five levels that serve the purpose of showcasing the capabilities of the game.

ii. **Introductory material**
- Sirah wakes up in the first area of the game and starts exploring the world. The area puzzle must be completed in order to progress to another level.

iii. **Mission objectives**
- The player must collect Soul Points in order to manage to start the puzzle and complete it.

iv. **Physical description**
- A simple 2D platformer world system filled with a dark, yet scenic view. Trees, Mountains.

v. **Map**

vi. **Enemy types encountered in-level**
- There are no enemies in the first level. This level focuses mainly on exploration and puzzle solving.

vii. **Abilities/power ups available**
- Wing Jump

viii. **Level walkthrough**
- The player begins in the middle of a plain and must proceed to the right where they find a platform they must jump tol.

ix. **Closing material**
- In the end, the player arrives to the next section/level of the game.

i. Level #2

i.Synopsis
- The game is divided in five levels that serve the purpose of showcasing the capabilities of the game.

ii. **Introductory material**
- To be defined.

iii. **Mission objectives**
- Collect Soul Points and complete the puzzle.

iv. **Physical description**

- An underground bright place filled with water sheets and flowers.

v. **Map**

vi. **Enemy types encountered in-level**

- Dark Goo (Depression remnants)

vii. **Abilities/power ups available**

- Power Flight

viii. **Level walkthrough**

- To be defined.

ix. **Closing material**

- To be defined.

j. ***Level #3***

i. **Synopsis**

- The third and final level of the game. The final boss fight takes place here.

ii. **Introductory material**

- Depression introduces itself to Sirah and tries to kill her.

iii. **Mission objectives**

- End depression.

iv. **Physical description**

- Large place that has floating platforms that can damage Depression.

v. **Map**

-

vi. **Enemy types encountered in-level**

- Dark Goo

vii. **Abilities/power ups available**

-

viii. **Level walkthrough**

- To be defined.

ix. **Closing material**

- To be defined.
-

10. SECTION VIII: CUTSCENES

k. Initial Cutscene

i. List of actors

- Sirah, People

ii. Description of setting

- A crowded city. Sirah is on top of a building ready to jump off. Below her feet, cars and people going through her day can be seen.

iii. Storyboard thumbnails and Script

ii. Final Cutscene

iv. List of actors

- Sirah

v. Description of setting

- Hospital, Hospital Bed.

vi. Storyboard thumbnails and Script

11. SECTION IX: GAME MODES

l. *Single-Player*

12. SECTION X: ASSET LIST

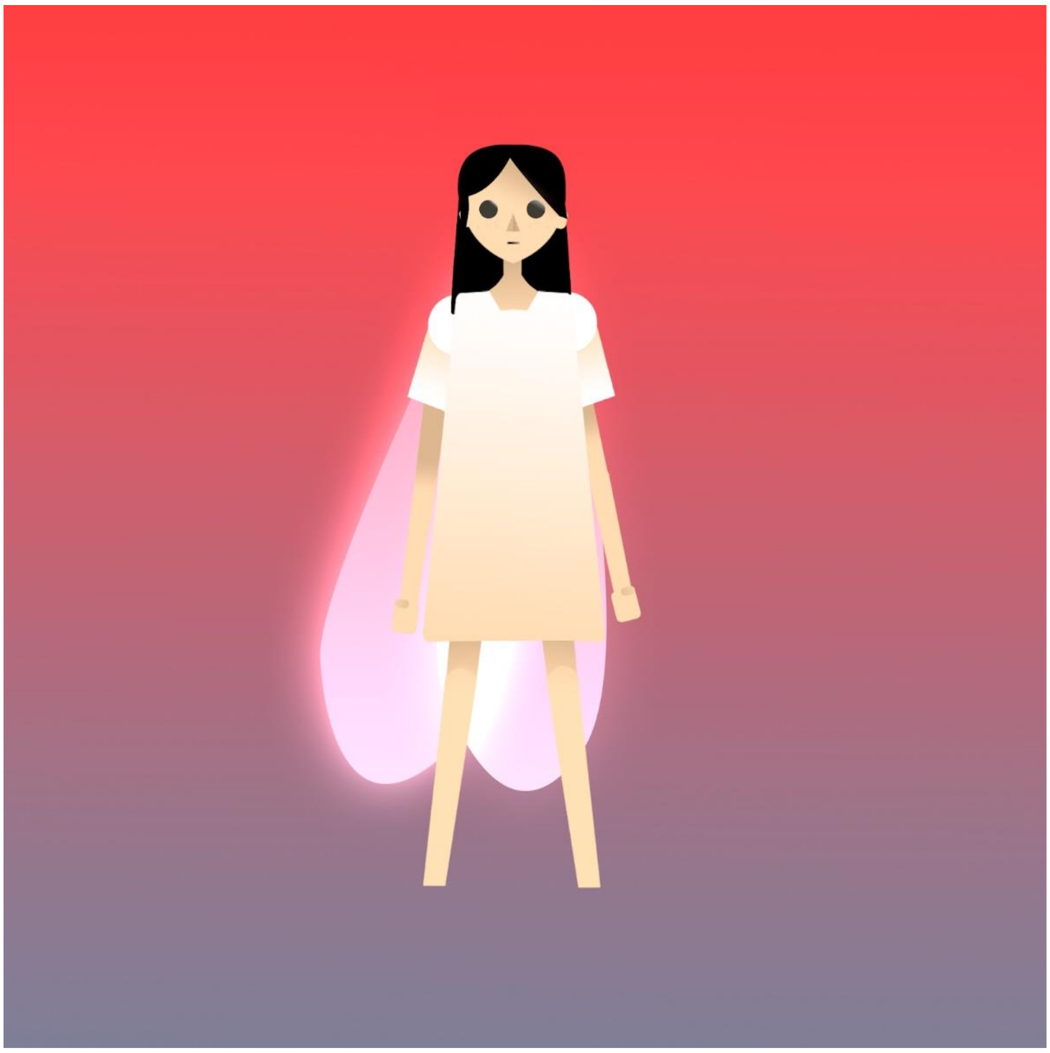
m. Art

i. Sprites

1. **Characters**

a. Sirah - First Draft:

Sprite that is being used in-game currently:



b. Depression:

2. Environmental Objects

a. Trees
b. Flowers
c. Grass
d. Sky
e. Bushes and other plants
f. Water Sheets

ii. Animation list

1. Characters

a. Sirah

i. Jump
ii. Walk
iii. Run
iv. Idle
v. Crouch

vi.Taking damage

b. Depression

.Idle

2. Abilities

. Sirah:

.Wing Jump

i.Light

ii.Power Flight

a. Depression:

i. Power Flight

ii. Floor Stomp

iii. Scream

3. Destructible or animated objects in the world

a. Various animals, trees, flowers, water, fireflies

iii. Effects list

1. Abilities effects list

a. Sirah's Wing Jump

b. Sirah's Light

c. Sirah's Power Flight

d. Depression's Floor Stomp

e. Depression's Power Flight

f. Depression's Scream

2. Environmental effects

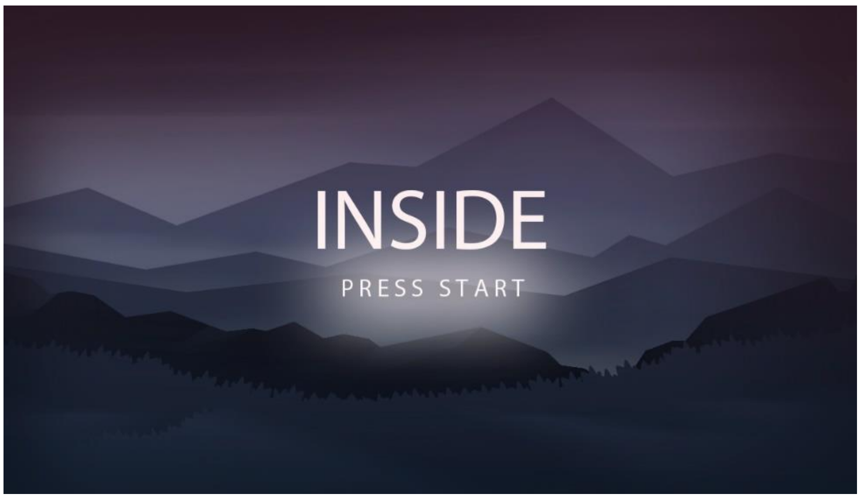
. Wind

a. Clouds

b. Dots of Light

c. Dust

iv.Inteface Art List



1. Icons
    - Buttons
    - New Game
    - Continue Game
    - Options
    - Controls
    - Exit

2. Menus
    - New Game
    - Continue Game
    - Options: Sound; Graphic; Resolution; Fullscreen; Field of View
    - Controls
    - Exit

n. Sound

i.Environmental Sounds

1. Sirah's walking/running sounds on different surfaces
2. Birds and other creatures
3. Wind
4. Leaves floating with the wind
5. Dust

ii.Abilities Sounds

1. Wing Jump

2. Power Jump

3. Light

iii. **Interface Sounds**

1. Various clicks, beeps, etc., as the player maneuvers through the menus.
2. Alert/acknowledgment sounds as the player picks up objects or his game state changes.

o. Music

iii. Ambient

1. Inside intro - 1 minute

ii. "Defeat" loops:

- Defeat loop + 5 seconds

iii. Cutscene music

1. Intro Music

a. A calm and sad piano starts the music. Violins and other ambient sounds gather.

b. Duration

- Close to 3 minutes

p. Voice

i.Actor #Sirah

1. Jump Sound
2. Receiving pain/damage sound
3. Interaction Queue

ii.Actor #Depression

1. Scream/Screech
2. Receiving pain/damage sound

13. SECTION XI: REFERENCES

a. Games

- Journey (2012)
- Rayman Legends (2013)
- Gris (2018)

b. Music

- Nils Frahm
- Gustavo Santaolalla