



TELECOMUNICACION

Campus Sur
POLITÉCNICA

ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA Y SISTEMAS DE TELECOMUNICACIÓN

PROYECTO FIN DE GRADO

TÍTULO: Diseño e implementación de la historia, las mecánicas y el flujo del videojuego serio "Phiby's Adventures v2"

AUTOR: Haoru Qin

TITULACIÓN: Grado en Ingeniería de Sistemas de Telecomunicación

TUTOR: Martina Eckert

DEPARTAMENTO: Ingeniería Audiovisual y Comunicaciones

VºBº

Miembros del Tribunal Calificador:

PRESIDENTE: María Luisa Martín Ruiz

TUTOR: Martina Eckert

SECRETARIO: Enrique Rendón Angulo

Fecha de lectura: 24 de julio 2020

Calificación:

El Secretario,

Acknowledgement

This moment, when I am finally going to finish this long journey in the UPM, a lot of faces come to my mind. Over these last years, I have been frustrated and lost, but I am also lucky enough to have so much love and support that helped me go through all the difficulties.

Here, I would like to thank all the professors I have had classes with in the UPM, all the classmates and friends. A very sincere thank-you to my tutor of this bachelor's degree project Martina Eckert, who has always believed in me, given me a lot of complementing and affirmation in my work, fully helped me in this urgent situation, and is always responsible. To my colleague Fernando, who is always enthusiastic, positive, and very helpful with all he knows (and he really knows a lot!) without complaint.

I would also like to thank all my family and friends who have loved me unconditionally. Among them, special thanks to my mom Ruiqiong, who loves me the most in the world and is always supportive. And another special thank-you to my boyfriend Jin, who has been my best friend, life tutor and the person who understands me the most in these last two and a half years. To Ra, without whom I would not have a story that I love so much. To Nana, for being the most adorable kitty in the world who is always there with me during this horrible quarantine experience.

All my experiences in university tell me that no fruit comes without effort. I am glad that I am finally tasting my sweet fruit and enjoying what I have learnt, not only the knowledge in Telecommunications area but also the life courses I have had with all the failures and successes.

Last, I want to thank whoever is reading this thesis, which I have been writing days and nights during the last month, thank you for making my writing meaningful and I hope it can be helpful to you as well.

Abstract

This document describes a part of the process of developing the serious game for physical rehabilitation purpose “Phiby’s Adventures v2”. This videogame aims at helping the patient (the player) to carry out the rehabilitation process with motion detection by connecting the game to Microsoft Xbox Kinect via a middleware K2UM. The game results will be used as feedbacks and sent back to the therapeutic web “Blexer-med”, which therapist set the parameters for the exergames to adapt to each patient.

The main software used for development is Unity whose scripts are written in programming language C#. A fully developed middleware K2UM is used to connect the hardware for motion detection Microsoft Xbox Kinect 2.0.

Based on existing features, this project is centered on increasing the playability by planning, designing, and implementing a storyline for the game, a few new mechanisms to improve the gaming experience.

Several basic and common game mechanisms are implemented in this project: the energy system, the dialogue system, and the inventory system. These mechanisms help both the player and the therapist to control the quantity and the difficulty of the exercises the player will be doing, being more familiar with the gaming environment and getting more engaged with it. An encrypted game data saving system is also implemented so the game data can be saved safely in local directories.

Considering that the game requires a lot of movement, other features for developers and therapists have also been implemented to make the game easy to test. A corresponding keyboard mode with exergames simulator is created so the entire gameplay can be done in keyboard mode. A new menu scene with the ability to select checkpoints is also implemented for test purpose.

As a result, the chapter one of the story created for this game is complete in this project, with 7 checkpoints implemented. Four exergames are designed to be in this chapter while two of them can be accessed and tested.

Resumen

Este documento describe una parte del procedimiento de desarrollo del juego serio con fines de rehabilitación física "Phiby's Adventures v2". Este videojuego tiene como objetivo ayudar a pacientes (jugadores del juego) a llevar a cabo la rehabilitación con detección de movimiento realizado por la cámara Kinect de Microsoft Xbox, la cual está conectada a través de un middleware llamado K2UM. Los resultados del juego se utilizan como feedback y se envían a la web terapéutica "Blexer-med", donde los terapeutas ajustan los parámetros para que los ejercicios se adapten a cada paciente.

El software principal utilizado para el desarrollo es Unity, cuyos scripts están escritos en lenguaje de programación C#. Se utiliza el middleware K2UM completamente desarrollado para conectar el hardware Microsoft Xbox Kinect 2.0 para la detección de movimiento.

Basado en las características existentes, este proyecto se centra en aumentar la jugabilidad mediante la planificación, diseño e implementación de una historia para el juego, y algunos mecanismos nuevos para mejorar la experiencia de juego.

En este proyecto se implementan varios mecanismos de juego básicos y comunes: el sistema de energía, el sistema de diálogo y el sistema de inventario. Estos mecanismos ayudan tanto a los jugadores como a los terapeutas a controlar la cantidad y la dificultad de ejercicio que realizará los jugadores, estar más familiarizado con el entorno de juego y estar más enganchado al juego. También se implementa un sistema de guardado de datos cifrados para que los datos del juego se puedan guardar de forma segura en directorios locales.

Teniendo en cuenta que el juego requiere mucho movimiento, también se han implementado otras características para desarrolladores y terapeutas para que el juego sea fácil de probar. Se crea un modo teclado correspondiente con un simulador de ejercicios para que todo el juego se pueda realizar en modo teclado. También se implementa una nueva escena de menú con la capacidad de seleccionar checkpoints para fines de prueba.

Como resultado, el capítulo uno de la historia creada para el juego se ha completado en este proyecto, con 7 puntos de control implementados. Cuatro ejercicios están diseñados para estar en este capítulo, mientras que dos de ellos se pueden acceder y probar.

Acronyms

GAMMA: Grupo de Aplicaciones Multimedia y Acústica

CITSEM: Centro de Investigación en Tecnologías del Software y Sistemas Multimedia para la Sostenibilidad

K2UM: Kinect to Unity Middleware

NPC: Non player character

HUD: Head-Up Display

Table of Contents

Acknowledgement.....	1
Abstract	2
Resumen	3
Acronyms	4
Table of Contents	5
1. Introduction.....	7
1.1. General introduction.....	7
1.2. Objectives of the project.....	8
2. State of the art	9
3. Design	12
3.1. General purpose.....	12
3.2. The storyline of the chapter one and the checkpoint system.....	13
4. Technical solution.....	16
4.1. Kinect and keyboard controls.....	16
4.2. The energy system.....	18
4.3. The inventory system	26
4.4. The dialogue system.....	31
4.5. The game objectives indicator.....	33
4.6. The checkpoint system in general	35
4.7. Part one: “The Cage”	38
4.8. Part two: “The Granny”	43
4.9. Part three: “The Bridge”	55
4.10. Main menu.....	67
4.11. Game data saving	76
5. Conclusion & future work	82
5.1. Conclusions	82
5.2. Possible future work.....	83

6. References.....	85
ANNEX I: Game flowchart with checkpoints in draft form.....	88
ANNEX II: Advancement Panel draft design.....	90
ANNEX III: Budget.....	91

1. Introduction

1.1. General introduction

Nowadays, as the technology develops in every area and the world starts to care more and more about the wellbeing of people with physical disabilities, therapeutic technologies are developed as well. Among all other achievements, movement detection technologies have been brought into the area of rehabilitation.

In this way, the research group GAMMA (Grupo de Aplicaciones Multimedia y Acústica) as part of the research center CITSEM (Centro de Investigación en Tecnologías del Software y Sistemas Multimedia para la Sostenibilidad) of the Technical University of Madrid (UPM) has been dedicated to this area during these last years. Several projects with therapeutic purposes for people with disabilities, mainly for children or teenagers, are developed or under development. These projects implement videogame environments for therapeutic rehabilitation by taking advantages of the motion detection that the Microsoft Xbox Kinect [1] offers.

Based on the project of developing the videogame “Phiby’s Adventures” using Blender [2], which started in 2014 and launched its demo version in 2018, this bachelor’s degree project is about the development and implementation of the second version “Phiby’s Adventures v2”, which first started in 2019. With the development of “Phiby’s Adventures”, a therapeutic web “Blexer-med” is developed as well, where “Blexer” comes from “Blender exercise”.

Before this bachelor’s degree project started, “Phiby’s Adventures v2” had already been partly developed with the protagonist, the terrain design, and a few other things. Being a new version of “Phiby’s Adventures”, it is controlled by Microsoft Xbox Kinect 2.0 and using the middleware K2UM, stands for Kinect to Unity Middleware, to receive and transmit data between the game and the web “Blexer-med”. But unlike the first version, “Phiby’s Adventures v2” is developed in Unity [3].

During the 2019/2020 academic year, this bachelor’s degree project is done within a working group including Juan Alberto García Fernández, who was also doing his bachelor’s degree project, Adrián Vázquez Chaves and Fernando Díez Muñoz, who worked as interns with Martina Eckert, the tutor of the project.

1.2. Objectives of the project

Starting with a partially developed version of “Phiby’s Adventures v2”, this bachelor’s degree project aims at adding the playability to the game. The main objectives are the following:

1. Adding mechanisms to make the player engaged with the game but not addicted to it in an unhealthy way:
 - Designing a story-based environment and letting the player feel the responsibility to advance the story and explore the unknown world.
 - Building other characters that give the player advises, accompany, encouragement and support so that the player can be more confident in the unknown future journey and not give up at the meantime being able to finish the tasks alone.
 - Designing and implementing a checkpoint system that divides the story into shorter parts and setting goals for each part to keep the player interested.
 - Designing and implementing an energy system which reminds or obligates the player to rest so the player does not overdo exercises.
 - Designing and implementing a game data saving system so the player can leave the game at any moment and return to where he/she has left, the next time playing the game.
2. Providing clear information about the progression and objectives so the player does not feel lost while playing:
 - Designing and implementing an inventory system which lets the player keep track of the obtained items and make use out of them.
 - Developing a dialogue system, so that the player can interact with other characters in the game and receive information about what to do next or get feedback of how well the player has been doing.
 - Creating an indicator for current objectives so the player is not discouraged in the middle of an objective for not knowing how much is left to do.
3. Improving other existing functions and creating new functions considering the convenience of non-patient players, such as therapists, health care personnel, and game developers:
 - Re-creating a menu with much more functions centered on game testing and debugging purpose.
 - Combining the two existing control modes, Kinect mode and keyboard mode, and making it easy to switch between them, so when testing, the control and operation can be chosen to be simpler.

2. State of the art

Speaking of gaming, the first thing comes into mind is entertainment. But as technology develops, video gaming is developed to serve more purpose other than pure entertainment. The term “serious game” is used to refer to these games.

In this project, Phiby’s Adventures v2 is a serious game whose purpose is to help people with physical disabilities to complete their physical rehabilitation sessions by gaming. Serious games as these are called rehabilitation games, which intend to help patients to enjoy their tedious and repetitive traditional rehabilitation therapy in order to get a better result from it. In the field of physical rehabilitation, repeating a single movement is compulsory, but could be too dull to complete. Video games, on the contrary, encourage players to repeat the same movements until they achieve their goals, and this can be extremely helpful to physical rehabilitation.

As a result of the need of movements detection, main platforms for rehabilitation video games are platforms with motion controls, such as Nintendo Wii, Microsoft's Xbox Kinect (which is our case), Sony's Eye Toy, and virtual reality. Most of them are affordable by most families and it makes video gaming rehabilitation even more practical, not to mention that patients will generally enjoy and feel more engaged in a gaming environment and less likely to take it as an obligation. And as smart phones and tablets are very common nowadays, some mobile phone applications also work as rehabilitation games, which is an absolute economic choice for not only patients but also therapists and developers.

Here are a few cases which are already in the market:

MindMotion™ Go [4] is a healthcare-professional-oriented gamified neurorehabilitation developed by MindMaze, a 2012-found spin-off from the Ecole Polytechnique Fédérale de Lausanne. It is also depending on Kinect for motion detection and control. It contains various types of games centered on different kinds of movements, and healthcare professionals can choose among them for certain patients.

MIRA [5], another software rehabilitation platform, uses motion tracking sensors developed by a team across the UK and Romania. It has more than 450 exergames which meet different requirements for different patients.

WalkinVR [6], with a little different concept among the first two cases, is oriented not only to people with physical difficulties but also healthy people who want to do workout in Virtual Reality (VR). It is a platform that can be bought by players directly from Steam, a PC game selling platform, and used for publicly available games. Gameplay in VR with assistance of another person with an Xbox Controller, virtual movement and rotation, adjustment of controller position and tracking of deficient or spastic hands are four main features that allow disabled people to be able to move in a certain way in VR which is not allowed in real life because of their physical limitation and therefore play real games that are not only designed for rehabilitation like a healthy person, and by doing this, achieve the goal of doing workout.

Another project PRIME-VR2 [7] is still in progress. It is funded by H2020 (Horizon 2020) and coordinated by the University of Pisa. It is also based on Virtual Reality and aims to create a platform that integrates bespoke VR controllers and physically- and socially-optimised gaming in order to deliver enhanced VR experiences. Another characteristic is that it takes social life into consideration. Players are in an online community where they are encouraged to play and compare with other users.

In addition to the cases mentioned before, some other games, which are not designed as a serious game, were also used to serve a purpose of rehabilitation. One of them, is the very popular early-smart-phone-age game “Fruit Ninja” [8]. Through feedback, therapists can track patients’ fingers and hands movements and judge how well they are doing the rehabilitation.

In the short history of rehabilitation gaming (first written evidences found in literature in early 1980s [9]), the results turn to be positive. Many studies and experiences prove that combining games in rehabilitation procedures makes patients to be more willing to do the exercises. A lot of patients even have more energy using rehabilitation games than in a traditional rehabilitation process, which is beneficial for most cases.

But on the other hand, there are some down sides. Occasionally, some feedback reports show that patients are more likely to get injured because of lacking the appropriate indication of therapists or unawareness of correct postures. Even though there are a few shortcomings, they do not stop video gaming rehabilitation from having a bright future in cooperation with therapists and may even take over the job of traditional therapists one day in a far future.

Before the development of Phiby’s Adventures v2 started, an older version called Phiby’s Adventures was done by 2018. In that version, Phiby could travel through different rooms by doing different types of exercises. There are four types of exercise: swimming, climbing, flying, and chopping trunks. Swimming exercise requires players to move their arms back and forth; climbing exercise requires players to move their arms up and down; flying exercise needs player to keep their arms stretched out for a period of time; and in chopping exercise players are asked to raise one arm and then put it down. All of them are different scenes that are loaded outside of the main scene, so these exergames are also known as mini games. The idea of having different exercise-scenes remains same for this new version.

The version 1 was controlled by the first version of Kinect (Xbox 360), and the middleware “Chiro” was designed to download and upload information and feedback to the web platform “Blexer-med”, whose name comes from “Blender Exergames” as Blender is the main designing platform. Therapists set parameters for different exercises according to each patient, and the game will download these parameters to adjust the mini games. After patients playing, feedbacks will be sent back to the web and therapists can read and analyze the results.

In Phiby's Adventures v2, the platform we use to detect movements and control the main character, Phiby, is Kinect. But it also includes the possibility to be controlled by keyboard, which allows an easier development of the game and to show and explain to therapists the gaming environment. The Kinect control is used for the players who are supposed to play the whole game without the help of keyboard control, i.e. it is used to make the game character walk around by slight corporal movements and to make it perform certain tasks that require stronger physical exercise, e.g. lifting the arms.

As figure 1 shows, the terrain of the game is an island with mountains on one side of its edge and contained a few zones that were separated by a Y-shaped river. In this new version, players are free to explore the whole island. At the beginning of this project, there were neither a story nor a background, just two mini games that were triggered on different spots on the island, which were different types of exercises a patient might be told to do.

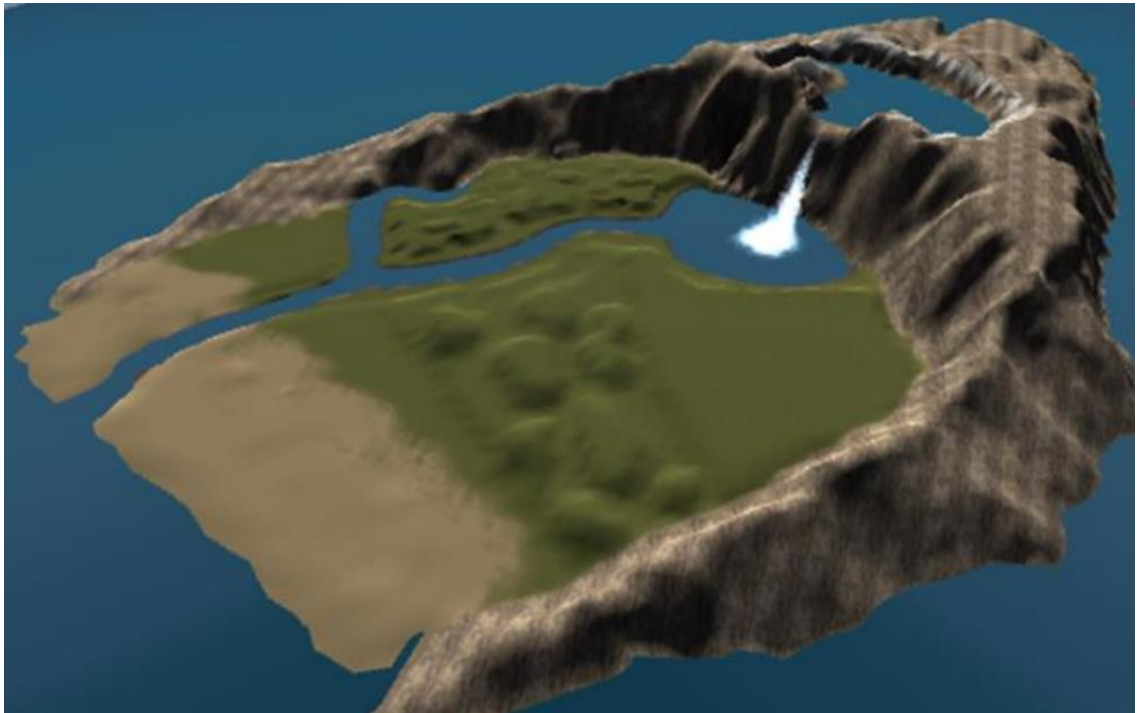


Figure 1. Y-shaped river and the terrain of the island [10]

At the spawn point of Phiby, near to a little cabin, a group of trees (different from the trees of the environment) was situated in front of the cabin, and a pile of wood trunks were put at one side of the cabin. Both were triggers for the mini game scenes, "Climbing" and "Chopping", respectively.

But as there was no story, players did not feel the need and desire to explore the island (which is one of the goals we want to achieve), and might even feel lost at the beginning, not knowing where to go nor what to do. Therefore, based on what we already had and what we needed to add, my work started.

3. Design

Before all the technical implementations, a good design comes convenient as a solid base of the game. In this chapter, a design is discussed, which keeps patients (or players) entertained and engaged during the rehabilitation process.

3.1. General purpose

Phiby's Adventures v2, as its name says, is an adventure game. But first of all, it is a so called "Exergame" (exercise game), whose main purpose is to help patients who have the need of rehabilitation to do physical exercises. The idea of being an adventure game consists in an attractive story that gives players goals and motivation to continue playing in order to do exercises in the process. Since it is a physical rehabilitation game, the amount of exercises a player should do is fixed by a therapist. For the purpose of maximizing the pleasure and engagement and minimizing the feeling of being in the middle of a rehabilitation, what a player should do, when and how much he/she should do it, needs to be reasonable and natural. The feedback is also supposed to be sent back to the therapist, so that the therapist can track if the patient has done well and how to readjust the rehabilitation plan for the patient if necessary.

The purpose of this project is to add a storyline to the existing island and characters. Players will play in a third-person perspective as the protagonist Phiby, explore the island and continue the adventures. A clear objective should be given to players so that they will be motivated to continue the game. Balancing between not doing enough exercises and over doing exercises, an energy system and checkpoint system is introduced. Players will have to rest between sessions of mini games if energy is low, which keeps players from over doing any of the exercises. But they also must finish the given goals to move on to the next part of the story.

The final aim is to insert all four types of exercises of Phiby's Adventures v1 into the new environment and use them in different scenes and for different tasks, i.e. "Climbing" can be applied in trees, mountains or other places. This project concentrates on including two of them, "Climbing" and "Chopping", the others will be integrated later. Various rivers and a large lake has been integrated in the landscape to provide different possibilities for the "Rowing" and "Diving" exercises, but these require implementation of underwater vision and the change of behaviors (from walking on the land to floating and swimming in the water) which were beyond the scope of the here presented project. Also, the mini game of "Flying" (same movements as "Diving") is not tackled here, as there the reasons for flying will be given at a later point of the story.

3.2. The storyline of the chapter one and the checkpoint system

To keep players interested in an adventure game, it is important to inform them about their progression and show them what to do next. The island is designed with different zones, separated by rivers, which can be used to divide into various chapters defined in this project. Chapter one is defined in detail, implemented, and tested internally.

3.2.1. General focus and structure

Before finishing all contents of chapter one, players are not allowed to cross the bridge, which is a key to the chapter two. For future works, more parts of the island can be unlocked by progression. Unlocked parts remain unlocked unless the story needs them to be locked again.

Each chapter, then, is divided into checkpoints. Game progress is saved according to each checkpoint. If a checkpoint is incomplete, upon restart the game and load the save file, players have to play from the beginning of that checkpoint. It is an effective way to present a short-term objective. At the end of each checkpoint, there are plots showing players what to do next. Although this project is only about the chapter one, a backstory is necessary as a solid base and a guideline for future development.

Not all the backstory is given at the beginning of the game, but as game continues, more and more information will be given to the player. Even a part of the backstory can be revealed on the final scene as to solve the puzzle placed at the beginning.

3.2.2. Backstory

Far from all other cultures on the planet, there is an isolated island, Breezeland, which is ruled by a dragon family. In this latest generation, there are a few of siblings (to be determined further on), each one has a different special ability and personality. Our protagonist, Phiby, is the youngest among them.

Days are peaceful on the island, but Ra, the eldest of all the siblings, has always been too ambitious to just be one of the rulers on this very peaceful island where nothing needs to be ruled. He wants to leave the island but there is no efficient power system that could support him during the long journey across the ocean. So, one day, he came up with an idea: hitting up a kind of crystal which can only be found in the lava to provide energy, which, with enough quantity, would be enough to power a ship for a long time.

As the smartest one among the siblings, Phiby calculates the possible result and it turns out that a lot of crystals are needed. But the crystals are what prevent the volcano from erupting. When taking out this big amount of crystals, the volcano would erupt and do damage to the entire island and its habitants. He does not approve the idea, and except Ra, all other brothers are supporting him since “Phiby always has the right answer”. Ra feels humiliated and gets

furious. He locks them separately on different parts of the island and ignores the result Phiby has calculated, planning on digging out the crystals.

At this point of the story, the player enters the game. The next part will briefly describe the storyline for the seven checkpoints (from 0 to 6) of this first chapter, the objectives, and the exercise scenes (separated from the island) accordingly, if there are.

3.2.3. Description of the checkpoints

Checkpoint 0

Phiby wakes up in a locked cage where the only way to leave is by climbing up the bars and pass through a little gap between two horizontally set bars. But to climb up, the bars are so slippery that Phiby will keep falling off. There is a cart with straw, which is supposed to be a place to rest. Phiby is clever enough to quickly have the idea of picking up those straws and attach them to his palms, so he can easily climb up now. After successfully finishing the mini game, checkpoint 0 is complete.

Objective for this checkpoint: Finding a way to get out of the cage

Mini game: Climbing up the bars of the cage

Checkpoint 1

Not very far away from the cage, there is a little wooden hut which belongs to the grandma of the siblings. Phiby decides to visit her first for possible information about other siblings before going anywhere else. At the time Phiby meets the grandma, she tells him that she is craving some apples that she has inside the cabin, but the key is lost. Phiby must find the key to be able to access the hut and once inside, finds out that all apples are eaten up. He now goes back to the grandma and is told to get some apples for her by climbing the apple trees in the garden next to the hut.

Objective for this checkpoint: Finding the key to the cabin

Mini game: None

Checkpoint 2

Phiby climbs the trees to get apples. Then he puts the apples to the bowl in the cabin. The grandma is very glad and starts saying that she saw that the bridge leading to the other part of the island is broken for some unknown reason. Phiby goes to check the bridge.

Objective for this checkpoint: Getting apples from the apple trees and putting them into the bowl

Mini game: Climbing apple trees

Checkpoint 3

The bridge is almost completely broken. Only a few wood structures are still visible. But the bridge is necessary to cross the river and reach the volcano, as Phiby cannot swim. But luckily, there are a few coils of rope on the ground next to the bridge. He tries to throw these ropes to the other side of the bridge to make it be able to support some planks as the floor of the bridge, but he fails.

Objective for this checkpoint: Trying to throw ropes to partially repair the bridge

Mini game: Throwing rope with failure

Checkpoint 4

Phiby is thinking of Rocky, his super strong brother. If he finds him, he can copy his ability and have enough strength to achieve the goal. This also reveals the special ability of Phiby: he is able to copy special powers of the other siblings, when they are close to him. Being unable to do anything else, Phiby starts to explore. Animals he meets tell him that they have seen Rocky near the cave. Phiby approaches the cave, and finally finds Rocky. Rocky is trapped between two huge rocks that he has to sustain, so he cannot move or otherwise the rocks will fall and damage him. As Rocky now is close to Phiby, Phiby obtains his super strength and can help Rocky to get out without hurting himself. After releasing Rocky, Rocky decides to follow Phiby.

Objective for this checkpoint: Finding Rocky and releasing him from the trap

Mini game: None

Checkpoint 5

Phiby goes back to the bridge with Rocky. He tries to throw the ropes again and now he has the super strength. This time, he is able to do it easily.

Objective for this checkpoint: Leading Rocky back to the bridge and using his super strength to successfully throw the ropes

Mini game: Throwing rope with success

Checkpoint 6

Now the only thing left to do is putting some planks on the ropes to completely repair the bridge. He remembers that near the hut, on the way to the bridge, there is a pile of wood trunks. He goes back to that place and chops some trunks and then returns to the bridge. Using the planks, he finally completes the bridge. Now he can cross the river through the bridge. Chapter 1 is over.

Objective for this checkpoint: Chopping trunks to make wood planks and completely repair the bridge

Mini game: Chopping trunks

4. Technical solution

4.1. Kinect and keyboard controls

4.1.1. Purpose

Phiby's adventure is a game designed to be played with Kinect. At first, for the convenience of developers, a different Phiby game object which was bound with a keyboard control script "animationPhiby.cs" was created by Miguel Ángel Gil in his bachelor's degree project [11], who also created the Kinect controlling script called "PhibyKeyboardController.cs". The keyboard control is also a good idea for easily explaining to therapists how the gaming environment is, what the parameters means to the game and how they should adjust them according to patients. But the two game objects were not synchronized and only one can be chosen to start the game, which did not allow therapists to switch between them in the middle of the game.

To achieve the goal of being practical for therapists, it will be better if there is only one single Phiby, and the control can be easily switched between Kinect and keyboard at any point of the game. In this project, the two game objects of Phiby have been combined into one, and with the "K" key on keyboard, a switch between the two types of control will be done.

4.1.2. Process and result

There are three main parts that need to be combined: the prefabs, the controlling scripts and the different cameras.

The keyboard prefab is the original Phiby and the Kinect prefab had been adjusted to a bluish green color to distinguish from the keyboard prefab. Apart from the color difference, the Kinect prefab had a child game object "KinectAsset" which was created in the bachelor's degree project of César Luaces Vela [12]. The main function of this asset is to transmit the movement data of the player detected by Kinect to Unity. So, to combine these two prefabs, the keyboard prefab was chosen to be the base of the mixed one since the color is the original color and "KinectAsset" is moved to be a child game object.

A new script "PhibyController.cs" was created to activate and deactivate "KinectAsset" and to set the "rigidbody", which is a default Unity class for controlling an object's position through physics simulation, to be kinematic when in keyboard mode. The Keyboard prefab was controlled by the character controller module, an integrated component for game objects, which is also deactivated when in Kinect mode and activated in keyboard mode via the script. At first, "PhibyController.cs" also activated and deactivated the two scripts according to the control mode the player has chosen. But later, with the help of Fernando Díez Muñoz, who

was doing an internship with Martina Eckert during the spring semester of 2020, two scripts were combined into one script called “PhibyMixedController.cs”.

By the fact that there were two cameras for these two game objects of Phiby, different scripts were used. In order to combine them, the script for the camera of Kinect Phiby was moved to the camera of the keyboard-controlled Phiby and a switch between them was added using the same script “PhibyController.cs”. Fernando Díez Muñoz helped to improve the motion quality of the Kinect camera, to smoothen the change of the point of view when switching and to combine the two scripts into one.

Before, the Kinect camera was a little more distanced from Phiby than the keyboard one, considering the practical use for Kinect requires more view than playing in keyboard mode. Also, the keyboard mode camera is controlled by moving the mouse, which allows players to even see the front side of Phiby, while the Kinect camera is fixed to look at the backside of Phiby. And these two reasons caused a disconnected view. After the improvement, every time the player switches from keyboard to Kinect, the camera will smoothly move from wherever the camera was looking at the last moment in keyboard mode, to the fixed position to look at the backside of Phiby. When Phiby starts to move in Kinect mode, camera will move a little farther away to give a greater view of the environment for the purpose of obtaining a better controlling experience.

Now a combined game object is used to present the protagonist Phiby and is switchable between the two modes at any moment of the game.

4.2. The energy system

The target players of this game are patients who need to do physical exercises. For the purpose of not overdoing and reaching fatigue, an energy system has been integrated. It aims at reducing energy with movements that are considered as exercises and increasing energy during the time when players are resting. With this energy system, players will have to regain energy by resting when they are considered tired or exhausted, to be able to continue the game.

4.2.1. Two values and four states

The system is designed to have two related energy values and four states of energy. The two values are the on-screen value and the potential value, which means an internally handled value, and the four states are “maintaining energy”, “decreasing energy”, “waiting-for-mushroom” and “enough mushrooms”. Mushrooms are game objects added to the main island scene for increasing the energy by picking them up. It will be further discussed in 4.2.3. with other features.

The on-screen value is always shown to the player in form of a line of mushrooms, whereas the potential value is not known by the player. During the state of waiting for a mushroom, the potential value increases constantly until it reaches the maximum of 100%, which is also the condition to switch from the “waiting-for-mushroom” state to the “enough-mushrooms” state.

The four states of energy are used to tell the system how it should react in different situation. In the main island scene, the state of decreasing energy does not exist. It is only triggered by entering a mini game scene (containing an exercise). In other words, the only state a player can have in mini game scenes is the decreasing-energy state. The other three states only exist in the main island scene.

For an easier understanding, in figure 2, a simple flowchart shows what conditions in each state must be fulfilled to switch to another state.

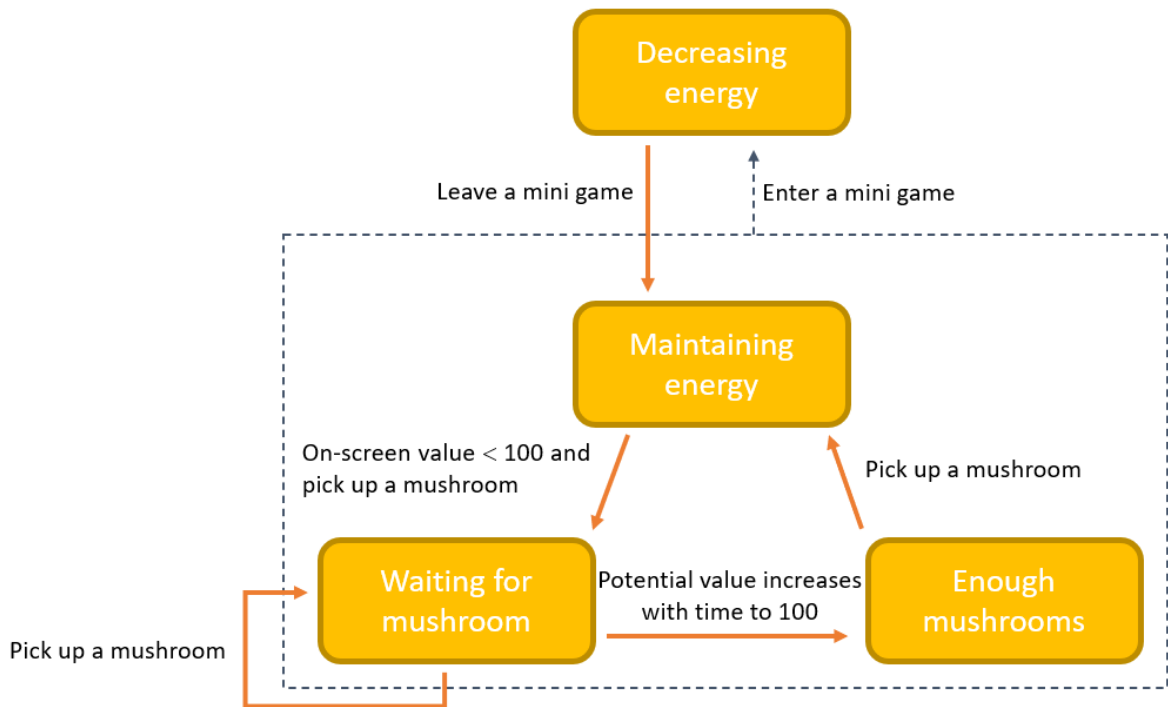


Figure 2. Energy states (yellow) and conditions of state switching

By entering the game from the main menu or leaving a mini game scene, the energy state will always be maintaining energy. Whenever Phiby has less than 100% on-screen energy, picking up a mushroom will switch the state to the waiting-for-mushroom state, which increases the on-screen value one point and the potential value starts to increase constantly with time. In this state, each time the player picks up another mushroom, the potential value will pass to be the new on-screen value. From the point of view of the player, energy increases with a random number by picking up mushrooms, while it is actually increasing with time and gets updated by picking up mushrooms. This loop ends when the potential value reaches 100% and the state changes to the “enough-mushrooms” state. The player will neither get notified of the state changing nor notice any difference. Only if a mushroom is picked up, will the on-screen value get updated to the potential value which in this state is always 100%. And by doing this, the current state changes back to maintaining energy.

At any moment of these three states, by entering a mini game scene, the current state will switch to the decreasing-energy state. Any potential value that have not been passed to be on-screen value will not count anymore, and the on-screen value shown by the energy bar at the moment of entering the mini game scene will be the initial on-screen value of this state. In this state, the on-screen energy decreases and passes to be the new potential value when the player leaves the mini game scene. Energy decreases with movement in mini game scenes and is calculated differently according to different formulas for each mini game. In this project, only one mini game scene “ClimbPhiby” has been designed with a decreasing energy formula and will be discussed later in the chapter 4.8.3. with more details.

Figure 3 shows another flowchart about how the two energy values work in different states. Blue text shows the different behavior of the two values in each state, and black text shows what happens to the two values at the moment of the state switching.

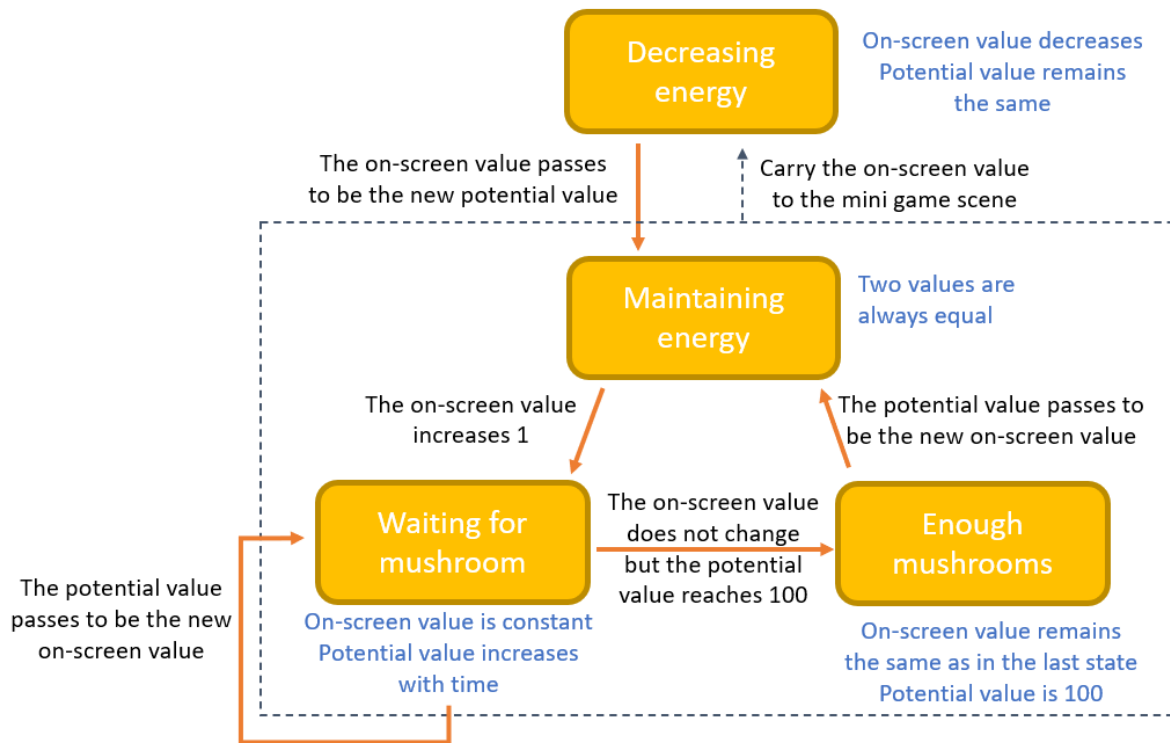


Figure 3. Relation of the two energy values and the four states

4.2.2. Visual presentation

The energy scale is set from 0 to 100 and visible through an energy bar showing at the upper-left corner of the screen, the number corresponds to the percentage of total energy, where 0 means no energy at all and 100 means 100% energy (full energy).

The energy bar consists of five mushroom icons [13], each representing 20% of energy. On the right side, a number is shown to indicate the energy value numerically. Figure 4 shows the energy bar when the energy is full, all the five mushroom icons are illuminated. Figure 5 is when the energy is 0, all icons become fully dark.



Figure 4. Energy bar with 100 energy



Figure 5. Energy bar with 0 energy

In between, every 20% of energy fully illuminates an icon, from left to right. When energy is not a multiple of 20, the rest that cannot be divided into 20, is shown by modifying the darkness of the icon. For example, if energy is 90, four icons will be lit, and the last icon will be half lit ($90 - 4 \times 20 = 10$, $10 \div 20 = 0.5$). Figure 6 shows how the darkness level decreases with energy from 0 to 20 (one icon representing range). The energy difference between each icon example is 2, i.e. 0, 2, 4, 6, ..., 20.



Figure 6. Mushroom icon darkness changes with energy

Energy levels can be manually changed through the Unity-inspector for easy testing. Two examples from the real game are shown in figure 7, for 25% of energy (20% energy highlights completely the first icon and the remaining 5% of energy highlights partially the second icon) and in figure 8, for 75% of energy (60% highlights completely the first three icons and 15% highlights partially the fourth icon). The manually changed on-screen value is marked by red squares.

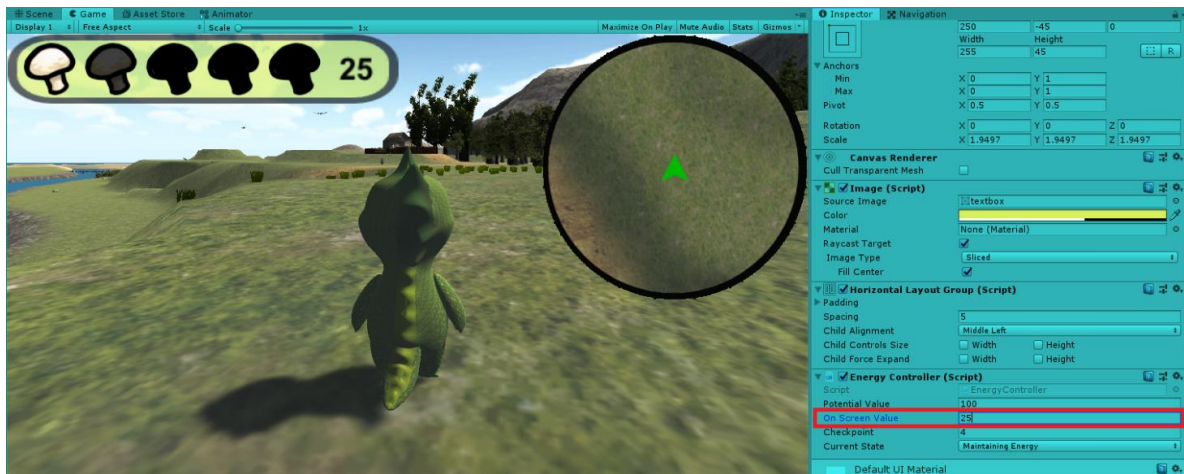


Figure 7. Visual presentation of 25 energy in real game

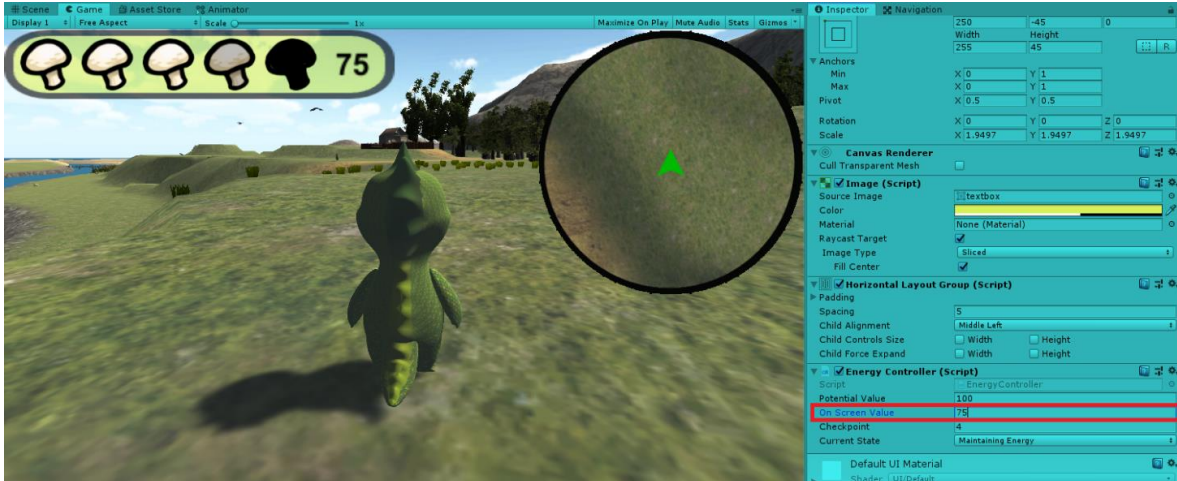


Figure 8. Visual presentation of 75 energy in real game

Although the minimum energy this energy bar can represent is 0, considering real life experience, 0 energy may lead to negative feelings and suggests “death”, so the minimum has been set to 10%. In the “Energy Controller” script, the on-screen energy value is protected from possible errors when manually changing number for testing purpose by a code to be in the range from 10% to 100%. Any number that is greater than the maximum will be set to 100%, and a number lesser than the minimum will be set to 10%. Notice that in figure 9, the on-screen value is manually set to -100, but the energy bar shows 10. And in figure 10, the on-screen value is manually set to 200, but the energy bar shows 100. In both cases, when hit enter after giving a number to the inspector, the value will be replaced by the number of energy shown by the energy bar, which means in the minimum energy case, that the on-screen energy will be replaced by 10, and in the maximum energy case, by 100.

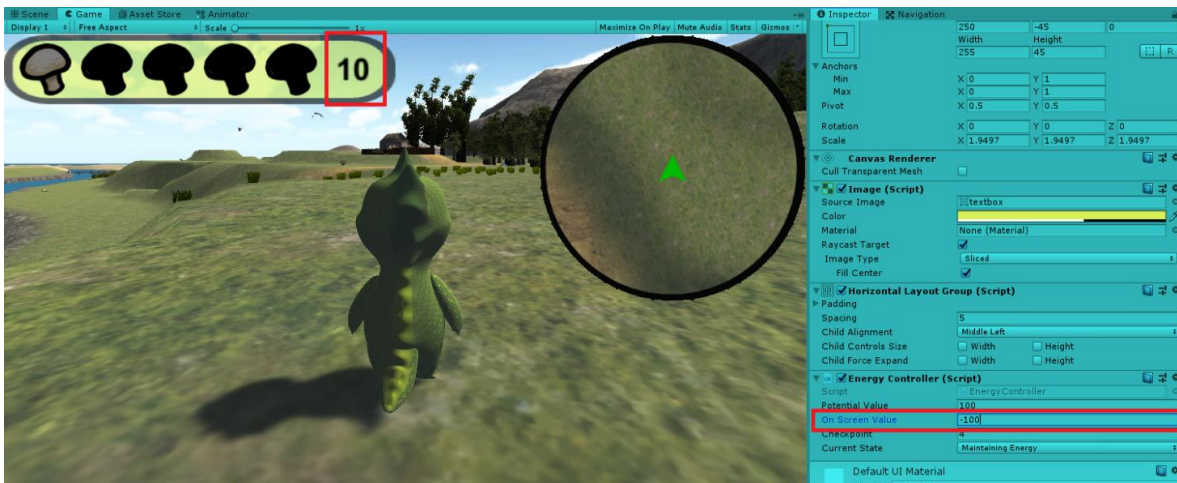


Figure 9. Example of the manually given on-screen value is lesser than the minimum

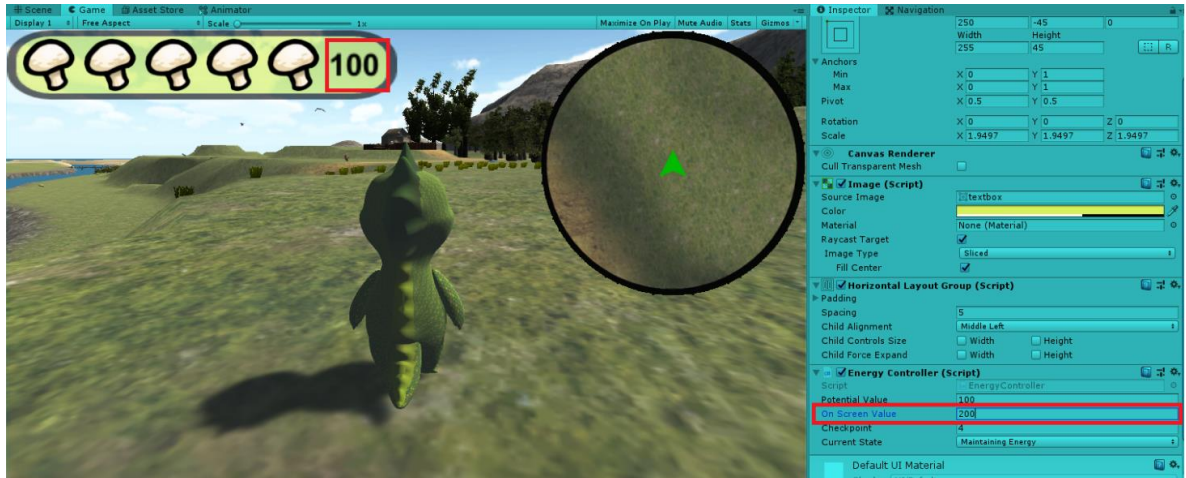


Figure 10. Example of a manually given on-screen value that is greater than the maximum

4.2.3. Mushrooms and their behavior

As mentioned in 4.2.1., players should pick up mushrooms to gain energy. When initiate the game, each mushroom chooses randomly one of the three prefabs [14] shown in figure 11, and at the correspondent location, the same mushroom icon used in the energy bar will be shown in the mini map.



Figure 11. Three possible prefabs for a mushroom

The mini map was created by Juan Alberto García Fernández[15] and uses a different layer, so that objects in the mini map layer will not be visible through the camera and players do not see these objects floating in the air. They are only visible in the mini map and are used to guide the player to the right place even from a greater distance. The mini map game object is located at the upper-right corner of the screen. In figure 12, circle 1 marks the mushroom game object and circle 2 marks how the mini map shows the player where the mushroom is. The green triangle mark represents the player's location in the mini map.

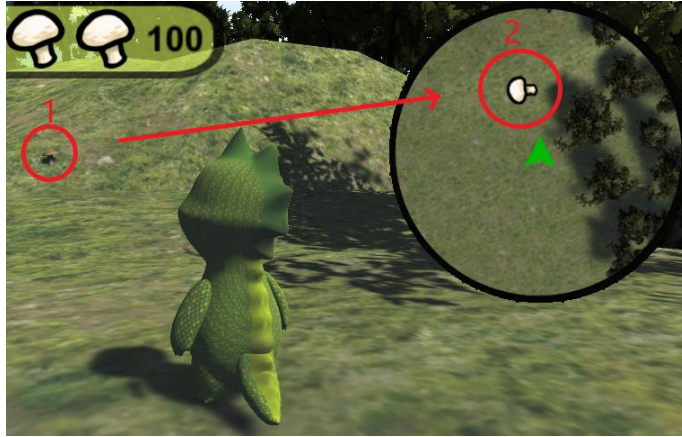


Figure 12. A mushroom near to Phiby and is shown in the mini map

When Phiby’s collider acts with the collider of a mushroom, the mushroom disappears. From the player’s perspective, this means that a mushroom has been picked up, players can gain energy according to the logic explained in 4.2.1. During the next 60 seconds after a mushroom is picked up, the icon of the disappeared mushroom becomes a little more transparent and disappears and reappears periodically with a period of 1 second. After these 60 seconds, a new mushroom grows at the same location of the picked mushroom with a random prefab. The 60 seconds is considered to be the regrowth time of mushrooms. Players can choose to take a rest by exploring the island and looking for mushrooms or staying at the same spot waiting for a mushroom to regrow. But after all, all that matters is the time players take to rest.

4.2.4. Mini games simulator for keyboard mode

Since the game is designed to be played with Kinect by default, keyboard mode is only an assistance mode. As the mini game scenes depend on movements for the exercises, they cannot be designed to be compatible with keyboard. But it will not be practical if the complete gameplay cannot be shown or explained in keyboard mode and has to switch to Kinect every time. So, in this project, a special scene named “mini games simulator” for testing purpose has been created.

In the island scene, when the player tries to enter the mini game scenes in keyboard mode, the mini games simulator scene will be triggered.



Figure 13. Mini games simulator scene

A general view of this scene is shown in figure 13. In the upper-left corner the energy bar is placed in the same way as in the main island scene. Since energy decreases in mini games, it also decreases here. The difference is that in this simulator scene the player uses the keyboard, and thus, the energy decreases with the space key to simulate whatever movement is required in the corresponding mini game. In the center, where we can see in figure 13 written as “Description Text”, appears a description of the scene and what to do according to different mini games. The purpose is to give a general idea about the respective mini game. The specific text for each mini game will be mentioned in later chapters. The Esc key is used to leave the mini games simulator manually without achieving the goal, which is also mentioned in “Description Text”. When loading the main island scene, the “Description Text” shows to the player that it is loading, as shown in figure 14.



Figure 14. Mini games simulator showing the main island scene is loading

4.3. The inventory system

In different parts of the story, tools or items may be required to solve puzzles or achieve goals to continue the adventure. Players need an indicator that shows what they have acquired and when they can use the items effectively. They can also get some hint from it when they have no clue about what to do next. An inventory is a place where to store tools and items the player possesses, and it is denominated as “Tools’ Panel” in this game. The name “Tools’ Panel” came from the idea of being a 2D panel in the HUD (Head-Up Display) canvas to store tools.

4.3.1. Visual presentation

The game object “Tools’ Panel” does not show anything, when there is no item stored. But when an item is stored, it appears in the middle of the right edge of the screen. It is adjusted to a little lower position to avoid the overlapping with the mini map. After considering the story, 6 slots are given to the Tools’ Panel, each slot can store one type of item. In figure 15, the relative position of the Tools’ Panel to the edge of the screen and the mini map is shown on the left side and the in-game visual presentation is shown on the right side, with all slots activated.

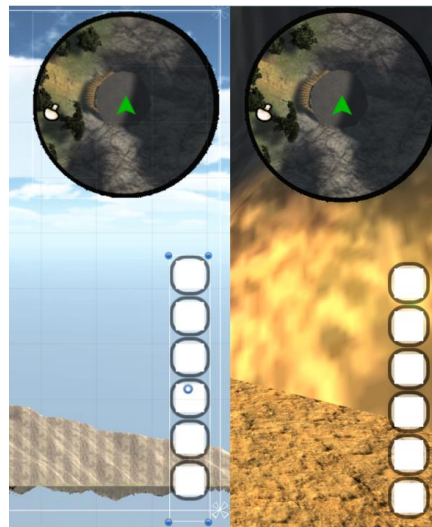


Figure 15. Relative position in editor (left) and in game (right) with all slots activated

The first slot has a fixed position, but the positions of the other slots change depending on how many slots are activated as shown in figure 16.

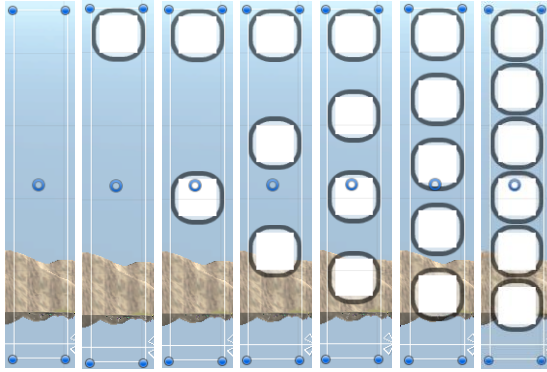


Figure 16. Alignment of slots when 0 to 6 slots are activated

One slot can contain various items of the same kind. When there is only one item, the icon associated to the item is shown in the slot, with no numeration needed. When there are more than one of a kind, a number is marked above the icon in a text type game object called “indicator” to indicate the amount. This number cannot exceed 99, which is reasonable and practical at this point. A blank icon example with an indicator of 99 is given as in figure 17.



Figure 17. Example of the indicator showing the amount 99

The items or tools to store can be classified into two types: basic items and consumable items (usually tools). The difference between these two types is that basic items will be used in different situations, and therefore will be saved in a file in between scenes and when exiting the game. But consumable items are items or tools that will help the player to achieve a goal in a short term and make progress. Usually, the story depends on whether these items have been achieved or not to be able to enter the next checkpoint, and therefore no information about them need to be saved between scenes.

4.3.2. Basic items (Apples and logs)

In the first chapter of the story, finished in this project, two basic items are defined: apples and wood. The apple-icon existed already in the mini map layer when there were apples on the ground. They are removed here due to lack of practical usage. The icon of a tree log is a copyright free image found on Pixabay [16]. In figure 18, these two icons are shown together and in the following order: original image, icon in the Tools’ Panel with only one item and the icon when there are two items.



Figure 18. Icons of apple and log and their in-game visualization

Apples and logs are proposed to have more usage than just in the two mini game scenes: “ClimbPhiby” and “ChopPhiby”, respectively. In fact, the two mini games are considered as the way to get them as resources for other goals. Once these mini games are unlocked (during the story), they become free to enter and the player is allowed to get apples and logs as much as he/she wishes and save them for later use.

4.3.3. Consumable items (Straws and key)

Different from basic items, consumable items are used right away or in a short time depending on the story. In this project, there are two consumable items: straws and a key. Both are highly related to game progression and play important roles in the game. Straws are needed to advance from the checkpoint 0 to the checkpoint 1 and the key is related to the grandma part of the story.

The straw icon is a screenshot of a new game object used in the cage of the checkpoint 0, and the key icon is a screenshot of an existing game object placed near to the hut of the grandma. The screenshots are processed into transparent-background images using a website called “Online PNG Tools” [17]. And the results are presented in figure 19 in the same order as in figure 18. As there is only one key in the game, the icon of key will never show the amount indicator in the Tools’ Panel.



Figure 19. Icons of straw and key and their in-game visualization

4.3.4. Programming logic and future usage guide

The behavior of the Tools’ Panel is programmed in a single script: “ToolsPanel.cs”. The number of slots is defined in the code, which can be modified as needed. Each slot has its own index from 0 to 5 (the highest slot with index 0 and the lowest slot with index 5) which is not visible to the player.

In the next paragraphs, the logics of some important methods in the script are explained.

```
public void RemoveFromToolsPanel(int index)
```

Parameter: `int` index - the index of the slot to remove

This method does not return anything. It is simply a method to deactivate a slot with an index, which is the integer passed as parameter. This method is useful for management outside of the script since the array of the slot game objects is not public and not accessible from other scripts.

```
public void SlotActivate(int num)
```

Parameter: `int` index - the index of the slot to activate

Similar to `RemoveFromToolsPanel`, no value is returned. The purpose of it is to activate the slot of a certain index from other scripts.

```
public int PickupNew(Sprite objectIcon)
```

Parameter: `Sprite objectIcon` - the icon in sprite format of the picked-up item

Return: index of the chosen slot or 7 in case of no available slot

This method is used when the picked-up item is the first one of its kind. As mentioned before, when nothing is stored in a slot, the game object of the slot is deactivated. Because of this, by detecting if a slot is active, we can know if it is empty. The icon passed as parameter will be put into the empty (inactive) slot with the smallest index number. For example, if the slot 0 and the slot 1 are active, the icon will be put into the slot 2.

This method returns the index of the first empty slot found, for easier management. No slot is inactive means there is no empty slot, and in this case, the method returns a preset integer 7 to distinguish from the others.

```
public int LookForItem(Sprite icon)
```

Parameter: `Sprite icon` - the icon in sprite format of the item to look for

Return: index of the slot when it is found or 7 in case of not found

This is a method used to look for if a target item already exists in the Tools' Panel. It returns the index of the slot if the target item is found in the Tools' Panel, otherwise it returns a 7, indicating that the target item does not exist in the Tools' Panel. The returned value is very useful for other methods.

```
public int ModifyAmount(int index, int operation)
```

Parameters: `int` index - the index of the target slot to modify

`int` operation - a code to indicate what is the operation to do

Return: the amount of the items in the operated slot

There are three kinds of operations that can be done with this method. When `int` operation equals to 0, one item of the same kind is added to the slot with the index passed as parameter. When `int` operation equals to 1, one item of the same kind is removed from the slot. When `int` operation equals to any other integer number, this method returns reading the indicator number of a slot.

There will be some special cases because when there is only one item of its kind in the Tools' Panel, the indicator game object is deactivated. By detecting if the indicator is active, the method will work a little differently. In case of the "adding item" operation, if the indicator is inactive, it means that there is only one item of the kind, and therefore it activates the indicator which shows "2". In case of the "removing item" operation, if the indicator reads "2", it deactivates the indicator game object which means one item to the player. If the indicator is already inactive, it will call the method `void RemoveFromToolsPanel(int index)` to remove the item, since it is the last one in Tools' Panel.

Despite how complex it might seem to be, it is very handy and simple to use these methods. If a future developer needs to expand the capacity (add more slots to it), it is easy to just duplicate the slot game object as many times as he/she needs and change the `const int numSlot` to the new number of slots. Assign the new slots to the array of slots in the inspector will do all the work.

For practical usage, one may find it handy to always look for an item using the method `int LookForItem(Sprite icon)` to get the slot index first. If it already exists and modify the amount by the method `int ModifyAmount(int index, int operation)`. Or else, when the returned slot index is 7, it should be handled with the other method `int PickupNew(Sprite objectIcon)` if the purpose is to store it in a new slot.

4.4. The dialogue system

A story-based adventure game contains a lot of conversations. There are a lot of ways to present dialogues. In this project, two different ways are used: audio dialogues and text dialogues.

When playing with Kinect, the player sits farther than playing with keyboard. This results in more effectiveness for keyboard mode than for Kinect mode because of difficulty of reading at a far distance.

4.4.1. Text dialogues

For better visual consistency, the background image used for the new game object “Tips Container” to present dialogues is the same as used for the energy bar. The name “Tips Container” comes from the original idea of that not only dialogues but also tips for players can be shown with this game object. It does not have its own script and all its functionalities are programmed in the GameManager.cs. GameManager.cs was originally created by Juan Alberto García Fernández [15], but here it has been hugely expanded because of all the new features created, especially for the checkpoint system. Since nearly all scripts are associated with GameManager.cs, and the text changing is a simple method, it is implemented in GameManager.cs for easier access whenever it is needed.

The method in GameManager:

```
public void ShowTip(string text)
```

Parameter: a text in string format, the dialogue or tip information to show on screen

It does not return anything as a result. It uses a enumerator `IEnumerator TipsShowing()` to make sure the text is shown for 2 seconds (modifiable) and then disappears. During these 2 seconds, no other text will be shown, which can prevent the situation where the second text is shown too fast that the first one cannot be read.

“Tip Container” is on the HUD canvas. It is fixed to the center-bottom of the screen. A real-game example is shown in figure 20.



Figure 20. Tip Container shows a dialogue where Phiby talks to himself

The text dialogues are in English because it is needed for international publications. In the articles for international publications, screenshots of the game would be posted. In future versions, it will be made selectable between Spanish and English.

4.4.2. Audio dialogues

Adrián Vázquez Chaves has brought his voice to the game by doing internship [18] with Martina Eckert, the tutor of this project, and created a lot of corresponding audio files to present all the existing dialogues in audio format. All audio files are recorded in Spanish, as the testing of the game will be made in Spain. In Kinect mode, audio files are played whenever there is a text dialogue in “Tip Container”.

4.5. The game objectives indicator

In some points of the story, there are missions to accomplish. At first, combining the dialogue system, the progression of the missions is told by other NPCs (Non-Player Character) through dialogues. But with a long-term goal that in the end text dialogues will be replaced all by audio dialogues, a clearer mission indicator is necessary.

4.5.1. Design

This indicator has been designed in form of a game object called “GameObjectives”. Being another element on the canvas, the position of “GameObjectives” related to the energy bar is shown in figure 21.



Figure 21. Position of “GameObjectives”

There are three important elements: the icon to represent the mission, the progression of the mission and the goal of the mission. By far, there are only two missions that use this indicator. One is the mission where the grandma asks Phiby to refill the bowl in the hut with apples, another is the mission where trunks are needed to repair the bridge. The icons of these two missions are shown in figure 22. More details about the logic of the missions will be described in the next chapters.



Figure 22. Game Objectives Indicator in-game visualization with the bowl mission (left) and the bridge mission (right)

Both icons are made in the same way as the straw icon and the key icon, which is mentioned in 4.3.3.

4.5.2. Functionality

As well as the dialogue system, the game objectives indicator does not have its own script but is integrated in the GameManager.cs. The progression of the mission and the goal of the mission are variables saved in a script called “gameSettingSustain.cs”, which was also originally created by Juan Alberto García Fernández [15]. It is used to save and transfer data between scenes and will be explained later with more details in 4.10.1.

```
public void ShowObjective()
```

This method is used whenever the “GameObjectives” needs to be refreshed. It sets the “GameObjectives” to be active, and depending on the current checkpoint, assigns the associated mission icon to the indicator. It accesses the “gameSettingSustain.cs” to get the current progression of the mission and the goal of the mission.

```
public void CreateObjective(int total)
```

Parameter: `int total` - the goal of the mission to be complete

When a new objective will be created, the only thing to do is to set up the total number of the item that the player has to gain during the process, which is the goal. An integer number is passed as parameter to be used as the goal. In this method, `void ShowObjective()` is called after the goal is set, so it will update the information to the indicator.

```
public void ProgressObjective(int newlyDone)
```

Parameter: `int newlyDone` - the number of the required item the player has gained newly

This method adds the newly done progression to the former progression of the player and gets a new number to represent the actual progression state. It also calls the method `void ShowObjective()` so that the indicator will be updated. For example: there are five apples in the bowl and the player adds three more to it. Then, the newly done progression is three and is passed as a parameter to the method. In this way “GameObjectives” gets updated and the progression of the mission is shown as eight ($5 + 3 = 8$).

```
public void ObjectiveComplete()
```

This method is used when a mission is complete. Calling this method will deactivate the game object “GameObjectives” and set the progression of the mission to 0 to initiate the value for the next mission.

4.6. The checkpoint system in general

The exact definition of checkpoint may be different for different games. Giant Bomb [19], a well-known American video game website, defines it as “a point within the game whereby the game saves its current state whether it be for the purpose of a more convenient respawn point or a gameplay design (such as in racing)”. In this project, a checkpoint is a point in time that allows the previous progress to be saved.

4.6.1. Purpose

There are two main purposes for implementing a checkpoint system for this game.

The first one is that with checkpoints, players will be more informed about the progression they have made. Combined with data saving in local files, players can save the game progress even after exiting the game. And the next time when they enter, they will start from the last checkpoint they have done instead of starting all over.

The other one is to give a diversity to the environment, which will change if the current checkpoint has changed. NPC may have different conversations, some restricted region may open to the player in a different checkpoint, a game object may react differently to an action in different checkpoints. This diversity will encourage the player to explore more, which is one of the goals of this project.

By applying a checkpoint system, a game can have much more possibilities when there is only limited space. In a future work, a visited zone may be totally different if that is what the story needs. And the checkpoint system will make this easier to implement.

4.6.2. General design and implementation

Checkpoints are closely related to the storyline. Therefore, it is impossible to limit it to a single script or a few checkpoint-system-only scripts. In fact, checkpoints exist everywhere whenever there is an action or reaction changing over different checkpoints.

However, the general variable to check the game progression, or in other words, to trace the current checkpoint is in “gameSettingSustain.cs”. It is denominated directly “checkpoint” and can be read and modified by other scripts. Its value will change if the set conditions are met. The very beginning of the story is coded to be the checkpoint 0. And the section between two checkpoints is defined as segments in this project. The relation between checkpoints and segments can be presented as in figure 23. In the figure, “cp” is used to represent the word “checkpoint”.

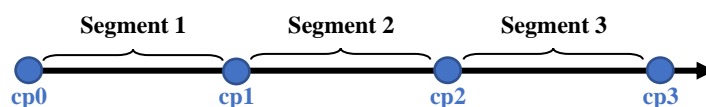


Figure 23. The definition of segments related to checkpoints

Every checkpoint is both the start point of a segment and the end point of another, except the first checkpoint and the last checkpoint, which can only be the start and the end, respectively. Segment n always starts at checkpoint (n-1) and ends at checkpoint n.

When implementing logics, switch statements are often used to give the possibility to react differently in different checkpoints. An example is given in figure 24, which shows a switch statement for a checkpoint indicating what to do in checkpoint 2, in checkpoint 6, and by default.

```
switch (checkpoint)
{
    case 2:
        objectiveIcon = bowlIcon;
        break;
    case 6:
        objectiveIcon = bridgeIcon;
        break;
    default:
        objectiveIcon = null;
        break;
}
```

Figure 24. Switch statement for checkpoint in the mentioned method `void ShowObjective()`

The variable “checkpoint” in “gameSettingSustain.cs” is given an initial value of 0. In “GameManager.cs”, this variable is initialized at the beginning of the execution in the start method - `void Start()`.

This can cause problems in some cases that the result of switch statements cannot be changed according to the checkpoint when the checkpoint value is changed, because it is only executed once when the method is called. As a simple solution, a Boolean-typed public variable `bool switchCP` and a related method `void SetCheckPoint(int cp)` are added to “GameManager.cs”.

```
public void SetCheckPoint(int cp)
```

Parameter: `int cp` - an integer number presenting the new checkpoint value

The variable `switchCP` is set to false by default in “GameManager.cs”. This method set its value to be true. `void Update()` is a default Unity method which runs once every frame. In other scripts where it is necessary to detect whether a checkpoint is changed or not, the value of `switchCP` should be detected in the update method to get notified when the value is changed to true, and therefore it can call the method which contains a switch statement for checkpoint. Also, the method with the switch statement should contains a line where the value of `switchCP` is set to false again, after it is called.

An example applying the logic explained in the previous paragraph:

Update()	ExampleMethod()
<pre> if (gameManager.switchCP) { ExampleMethod(); } </pre>	<pre> gameManager.switchCP = false; switch (gameManager.checkpoint) { ... } </pre>

For this reason, it is always better to set the checkpoint value with the method `void SetCheckPoint(int cp)` which can avoid possible problems.

To reasonably design where to set the checkpoints, it is important to make sure that the content between two checkpoints is appropriate in terms of length and effort required to complete. At the beginning of this project, a draft plan was made in form of a flowchart, which is attached at the end as ANNEX I. Despite of being a draft plan, it is very close to what is implemented in the end. Only a few subtle changes were made to ensure that the design is reasonable.

By the end of the implementation of the chapter one, there are totally six segments, which means seven checkpoints. When speaking of the storyline, logic, and spatial layout, some of the segments are highly independent from other segments, and some are inseparable from each other. According to the different focuses of the story, it is better to explain the implementation of checkpoints by dividing the chapter one of Phiby's Adventure v2 into three main parts: the cage, the Granny, and the bridge. All the different elements and features explained earlier - controls, energy, inventory, dialogue, objectives, etc. - are implemented alongside the checkpoint system to achieve a better result in playability. Next, the technical implementation of each part will be explained in detail separately.

4.7. Part one: “The Cage”

This part only contains the first segment, which means it starts at checkpoint 0 and ends at checkpoint 1. For this part, a huge cage is designed as all the activities are done inside of it until Phiby manages to get out and advances to checkpoint 1. Originally, it was denominated as cell. But taking into consideration that cell may cause negative feelings, it is renamed to cage, although occasionally it may be referred as cell, since the associated game object is named “Cell”.

4.7.1. Layout

Different from most other game objects in the game which are modelled with Blender or imported from the Unity Asset Store, the main structure of the cage is modelled directly in Unity. Figure 25 shows the prefab “Cell” from two perspectives.

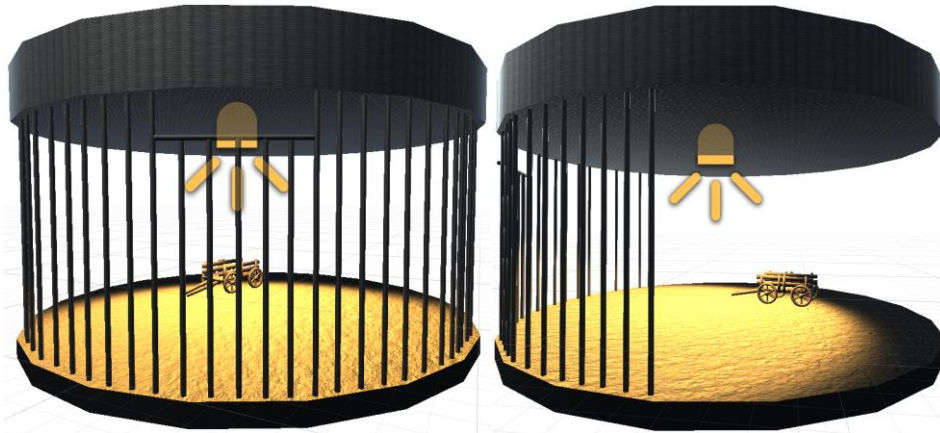


Figure 25. Front view (left) and right-side view (right) of the prefab “Cell”

The ceiling, the floor and the bars are all cylinders of different diameters and heights. The ceiling and the bars are given the same texture by using a metal material of the asset “Prototype Materials Pack” by Burak Taban [20], and the floor uses a dirt material of the same asset. A light source is added so the inside will not be too dark. There are no bars at the back side of the cage because in the game environment, the cage is partially embedded in the mountain, as seen in figure 26. Phiby is also seen in the figure at the point where the player spawns the first time in the game. Distances between every two bars are set and tested carefully so that Phiby’s collider is too large to allow Phiby to get out.

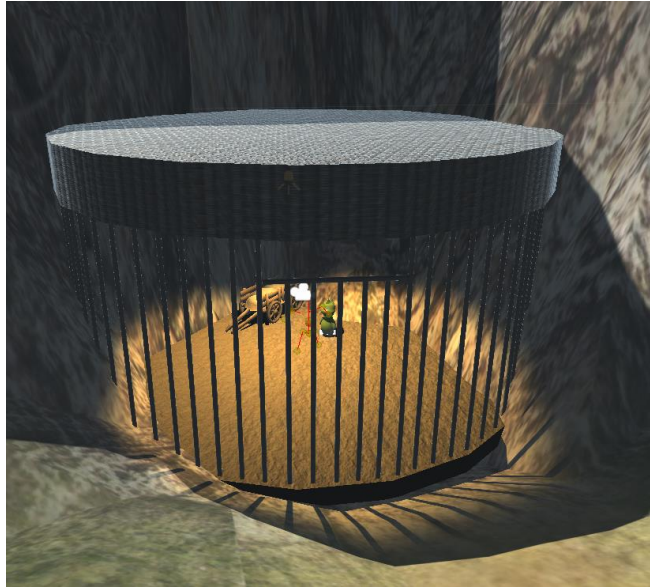


Figure 26. Prefab of “Cell” in the scene with the environment

In the cage, there is a cart, whose prefab is imported from the asset “Wooden Cart” by GrigoriyArx [21]. In the cart, there are three piles of straws whose prefab is also included in the same asset. The game object of a straw pile is named “Straw” and its screenshot is used to make an icon of straw, which has been explained in 4.3.3. It can be noticed that some bars are shorter than the others. The shorter bars form a separate game object named “ShortBars”. It is programmed that it can be triggered when it collides with Phiby. It reacts differently under different conditions. On the top of “ShortBars”, there is a gap between a horizontal bar and the ceiling. This gap is, according to the story, through where Phiby tries to get out.

4.7.2. Logic

The segment 1 starts at checkpoint 0 and ends at checkpoint 1. At the end of this segment, a mini game is involved. In this project, the mini game is only designed in concept without implementation and simulated with the mini game simulator for an uninterrupted game play experience. Therefore, there will not be a failure case for achieving the goal of the mini game. In future work, the Kinect mini game will contain an arm exercise, where the user has to “climb” up the bars.

A flowchart is designed to explain the logic of the segment 1 as shown in figure 27. It shows the logic from the beginning at checkpoint 0 and ends when the player enters the mini game. The value of the checkpoint is changed to 1 when the player achieves the goal of the mini game. More details about the mini game will be discussed in the next part.

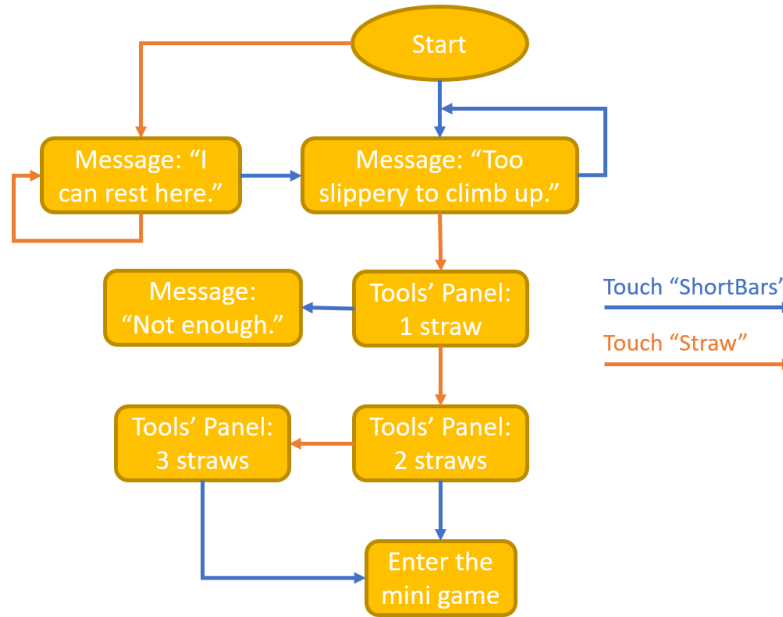


Figure 27. Logic design for the segment 1

There are arrows of two colors in the figure, each color represents an action. The blue arrows represent the action of Phiby colliding with the game object “ShortBars”, while the orange arrows represent the action of Phiby colliding with the game object “Straw”. All the messages shown in the flowchart are simplified. The corresponding phrase in the game are much longer and with much more details to avoid possible misunderstandings or confusion. With the help of the flowchart, the logic of the segment 1 is going to be explained.

At the start of the game, Phiby (the player) will spawn in the cage. At this moment, only two actions will get reaction: try to touch the short bars or try to touch the straws. Actually, touching the straws at the beginning is optional. It will show a unique text message, reminding Phiby that these may be used to sleep at night since they are soft and comfortable, but eventually, Phiby should get out, instead of staying in the cage forever. The essential action is to go near the short bars and try to get out by touching it. It shows the natural desire of the player that he/she does not want to be restricted in this little cage.

By touching the short bars, the player gets a tip to leave the cage through the little gap by climbing up. But Phiby needs something to help him climb up, which is also an indirect tip for the player to take some straws from the cart. And now, if the player tries to pick up straws by colliding with it, the straws can be picked up.

In the cart, there are in total three piles of straws. The player can choose freely how many piles he/she wants to pick up. But when there is only one straw in the inventory (Tools' Panel), a message will pop out saying that it is not enough. With two or more, a different message, showing the player that he/she is about to enter the mini game, pops out. After successfully

climbing up that bars and getting out, no matter how many straws there were in Tools' Panel before the mini game, at the beginning of the segment 2, no straws are there anymore.

4.7.3. Simulating the mini game

The mini game associated with this segment is designed but not implemented. It is designed to be an exergame of type climbing. It will be very similar to the one implemented for the apple trees near to the hut, but much shorter, which functions more like a tutorial. The background will be different from the apple trees mini game since the mini game takes place in the cage. The exergame of type climbing will be explained with more details in the part of the Granny.

To guarantee a continuous gameplay, the mini game simulator is used not only in the keyboard mode but also the Kinect mode. The mini game simulator will show different information depending on the control mode.

In keyboard mode, the text shows the information about how the mini game in Kinect mode is. As in figure 28 shows, the text works as brief introduction to the mini game and tells the player to press the Esc key to go back to the main island scene. Since the associated mini game has not been created yet, there is neither energy changing nor information for therapist, which for an existing mini game it means a guide to the parameters and their meanings for the patient.



Figure 28. Showing informative text in mini games simulator in checkpoint 0 when enters in keyboard mode

Usually in Kinect mode, the mini game scene is loaded instead of the mini game simulator. But in this case, it is a temporary scene that will be used until the real mini game of climbing the bars is created. Basically, the text shows the same information as in keyboard mode, but in place of exiting the scene by manually pressing the Esc key, the game reloads the main island scene automatically using a timer. There is another additional information telling the

player that the mini game is still under development. The simulator for the climbing mini game in Kinect mode is shown as figure 29.

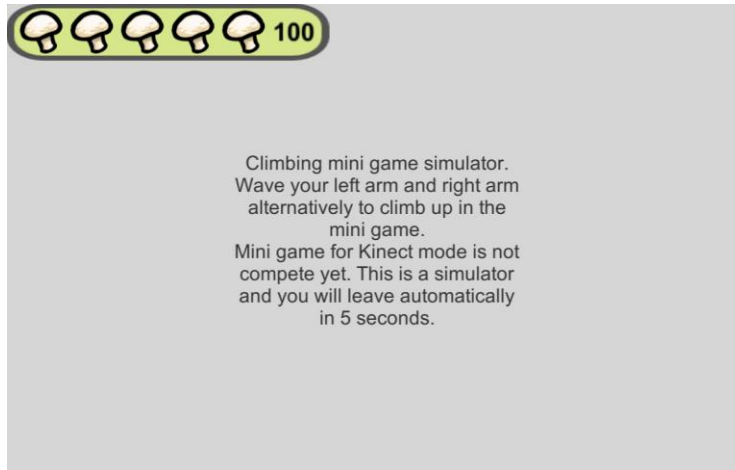


Figure 29. Showing informative text in mini games simulator in checkpoint 0 when enters in Kinect mode

4.8. Part two: “The Granny”

After a short experience inside the cage, the player is now outside the cage and free to explore the island. This part starts with checkpoint 1 and ends when the value of the checkpoint turns to 3. According to the definition, it contains two segments: segment 2 and segment 3.

4.8.1. Layout

This part of the story takes place mostly at the hut where the grandmother of Phiby lives. Before this project, there was another version of the grandmother. But due to the instability, it was replaced by a new model created with Blender, as shown in figure 30, by Fernando Díez Muñoz as part of his work during internship. After importing it into the Unity project, the prefab was named “Granny”, so she is going to be mentioned as Granny from now on.



Figure 30. Model of Granny

The hut, originally created by Cristina Esteban González [22] in Phiby’s Adventure v1 and modified the interior by Laura Molero Salazar [10] before this project, is relocated to be closer to the river and farther from the cage. The new location benefits from a more reasonable distances to the cage and to the bridge, an important place for part 3 of the story, “The bridge”. Figure 31 shows the former and the current location of the hut. The hut is also modified a little to improve the animation of the door, to resolve the problem of the lack of space, and to add some important new game objects including a bowl with apples and a transparent wall with collider. These are directly related to the new story logic.

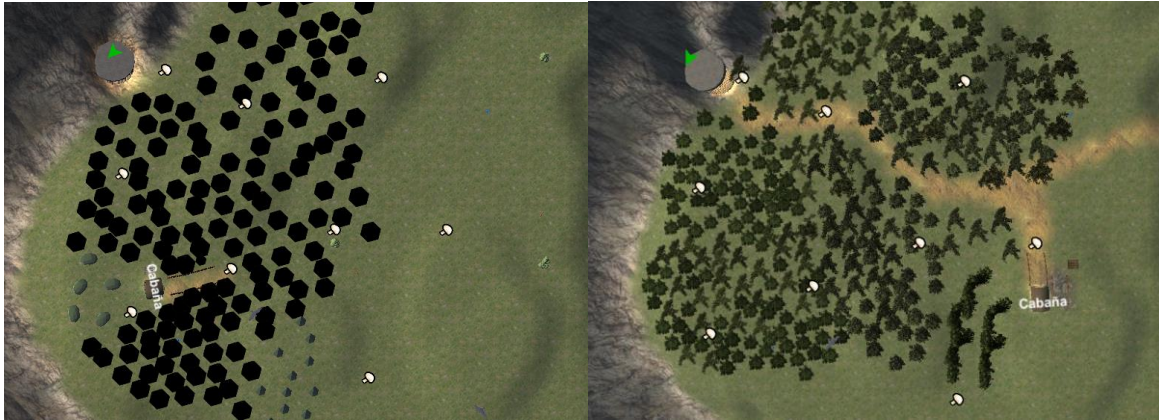


Figure 31. Locations of the hut before (left) and after (right)

As seen in the figure, a road is added to the island so that the player can follow the road and find the hut. The road continues to the bridge and other important places.

A garden with five apple trees where the mini game of this part takes place is also relocated. Figures 32 and 33 show the locations of the garden relative to the hut before and after the relocation, respectively. The garden is highlighted in the figures by a yellow rectangle. A noticeable difference is the texture of trees, which has been modified by Fernando Díez Muñoz to improve the visual performance.

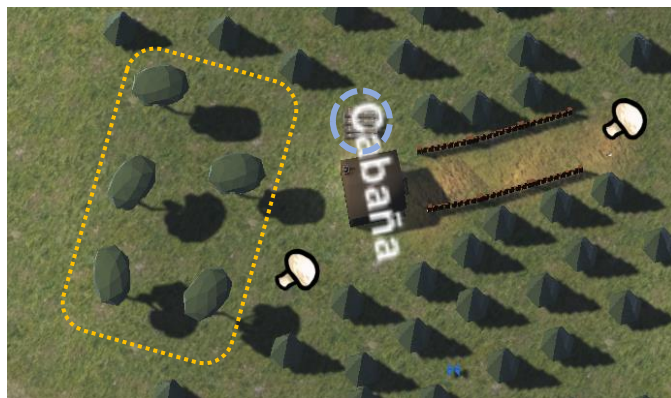


Figure 32. Location of the garden relative to the hut before the relocation

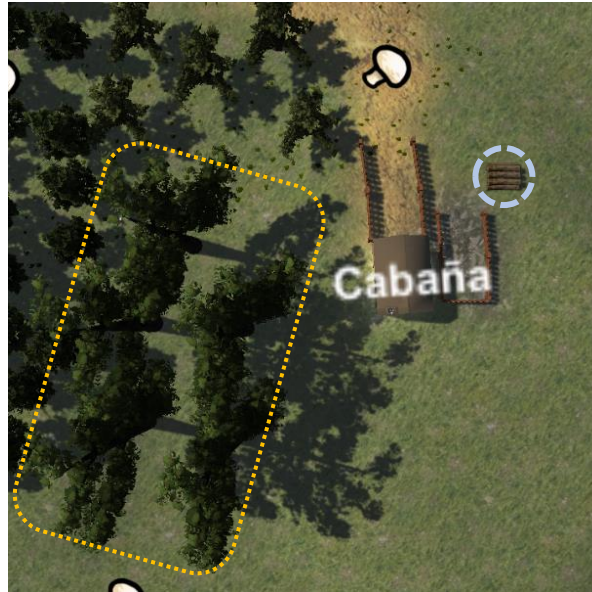


Figure 33. Location of the garden relative to the hut after the relocation

It can also be noticed that a pile of trunks is also moved with the hut, which is marked with a blue circle in figures 32 and 33. This pile of trunks is used as a trigger for the mini game “ChopPhiby”, the main exergame (mini game scene) in the bridge part. In this part, “ChopPhiby” is only a passive object.

Inside the hut, there are some pieces of furniture, most are purely decorative. A bowl on the desk in front of the door, as shown in figure 34, is the only functional object. It is used to visually present whether the objective of segment 3 is complete. The logic behind it will be explained in the next part.

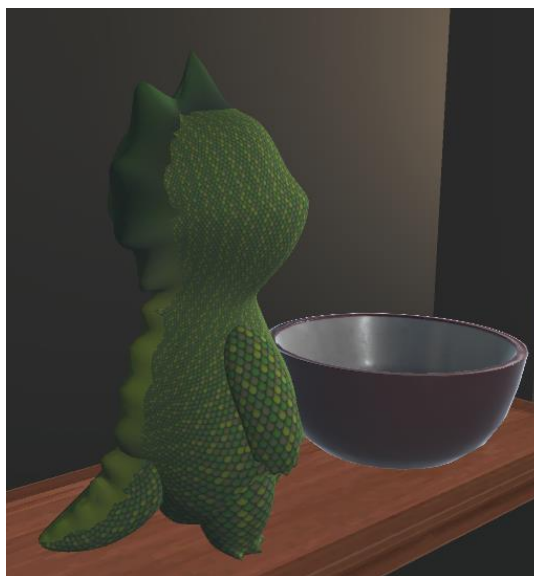


Figure 34. The bowl in empty state with Phiby on the desk

4.8.2. Logic

As this part contains two segments, it will be clearer to talk about them separately first, and then combine the whole implemented logic.

In figure 35, a flowchart is used to show the main logic of segment 2. Yellow arrows present the specific action of talking to Granny. The yellow rectangles are events that can be triggered by actions. There are two yellow rectangles with a dashed outline, which means that they are optional. The area surrounded by a dark blue dashed line means that at that moment, the door is unlocked.

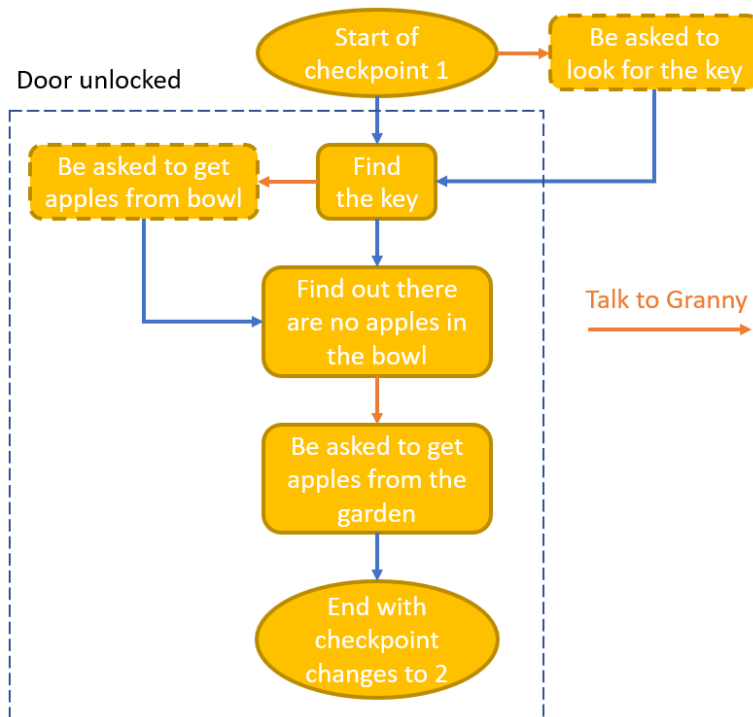


Figure 35. A flowchart presenting the logic of segment 2

Segment 2 starts when Phiby gets out of the cage. As mentioned before, the player can follow the road and arrives at the hut. A key is under a tree proximate to the road. The player must have the key to be able to enter the hut. This is asked by Granny if the player has still not found it (no key in “Tools’ Panel”). But if the player has picked it up before talking to Granny, the dialogue asking to look for key will not appear. And the state of the door also depends on whether the key is in the “Tools’ Panel”. The method `int LookForItem(Sprite icon)` in “ToolsPanel.cs”, mentioned in 4.3.4., made this easy to implement. Using this method to detect if the play has the key, then Granny and the door are able to react to a given situation correctly.

After entering the hut, the player has to interact with the bowl to find out that the bowl is empty. And at this point, by interacting with Granny, Granny knows that there are no apples

in the bowl and asks Phiby to bring more. When the conversation ends, a few changes take place: a suitable integer number is assigned as the goal of the objective for the next segment, the checkpoint value turns into 2, and the scene enters segment 3. The logic of segment 3 is more complicated than the logic of segment 2, because of the mini game “ClimbPhiby” and the objective associated with apples. A simplified flowchart used to explain the logic is presented in figure 36.

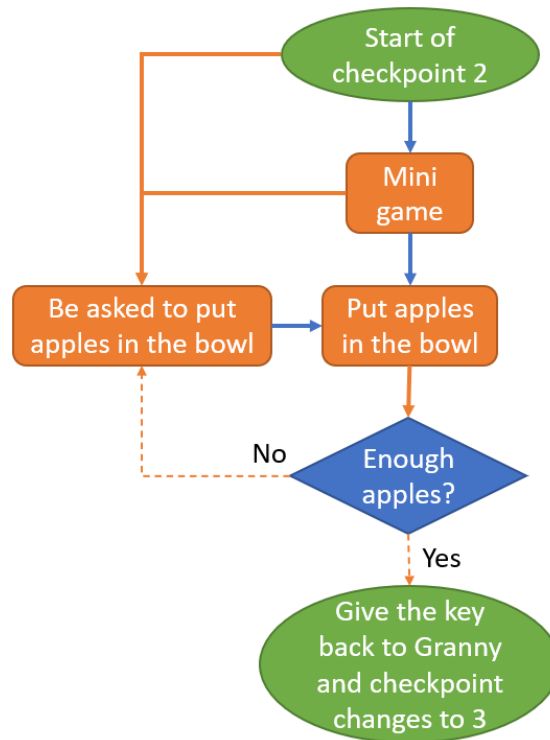


Figure 36. A flowchart presenting the logic of segment 3

The green circles present the beginning and the ending of this segment. The orange rectangles are events that happen in the process. The blue diamond shape means a condition to judge. The orange arrows are the process of interacting with Granny, the dashed ones present the automatically happened process after the judgement. The blue arrows mean interaction with specific objects that are not Granny.

Generally, the player has to get the apples from the mini game “ClimbPhiby”, and then put them into the bowl to complete the objective. The logic behind the mini game to obtain apples is explained in 4.8.3. with details about the rules for energy and some associated parameters. Here, the logic to judge the progression of the objective is discussed in a detailed description.

The algorithm uses five parameters: `int totalNumApples`, `int totalObjective`, `int objectiveDone`, `int totalApplesNeed`, and `int bowlApples`, all of them are saved in the “gameSettingSustain.cs” for easier data saving.

- `int totalNumApples` is the number of apples the player has in the “Tools’ Panel”.

The next two parameters are basic parameters for the “Game objectives indicator”:

- `int totalObjective` indicates the goal of the objective, in this case, it is the total number of apples the player needs to put in the bowl.
- `int objectiveDone` indicates the progression of the objective, in this case, it is the number of apples already in the bowl.

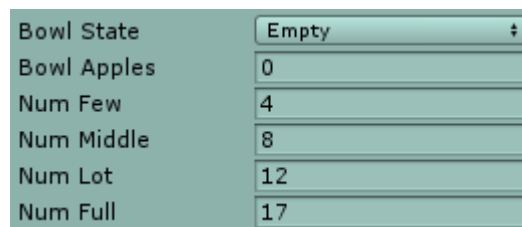
The last two parameters are especially created for this segment:

- `int totalApplesNeed` indicates the number of apples still needed to complete the objective.
- `int bowlApples` specifies how many apples there are in the bowl already.

There are five states for the bowl, determined by the value of `bowlApples`:

1. Empty: $\text{bowlApples} > 0 * \text{totalObjective}$
2. Few: $\text{bowlApples} > \frac{1}{4} * \text{totalObjective}$
3. Middle: $\text{bowlApples} > \frac{2}{4} * \text{totalObjective}$
4. Lot: $\text{bowlApples} > \frac{3}{4} * \text{totalObjective}$
5. Full: $\text{bowlApples} > 1 * \text{totalObjective}$

The limits for the last four states are calculated in the start method of “GameManager.cs” and saved in four parameters: `int numFew`, `int numMiddle`, `int numLot`, and `int numFull`. If `int totalObjective` is not a multiple of 4, the calculated results are rounded down to the nearest integers. An example is shown in figure 37, where `int totalObjective` has a value of 17, the four values are 4, 8, 12, and 17, respectively.



Bowl State	Empty
Bowl Apples	0
Num Few	4
Num Middle	8
Num Lot	12
Num Full	17

Figure 37. The numbers for bowl states when the goal of the objective is 17

At the end of segment 2, the goal of the objective is assigned. The number comes from the variable `int totalApplesNeed`, which means before putting any apple into the bowl, `int totalObjective` and `int totalApplesNeed` are equal. But with actions of obtaining apples from “ClimbPhiby” and putting apples to the bowl, the value of `int totalApplesNeed` varies while the value of `int totalObjective` stays constant.

It has to be mentioned that the original value of the parameter `int totalApplesNeed` is calculated with various parameters downloaded from the therapeutic web “Blexer-med 2.0”, passed to Unity via the middleware “K2UM”, and assigned in the start method of “GameManager.cs”. When there is no data downloaded from the web (using the middleware

without configuration), default numbers are assigned to these parameters, and `int totalApplesNeed` is calculated with these default values. This process, implemented by Juan Alberto García Fernández [15], ensures that the number is suitable for each player but also a little random. When the same player plays the game for the second time, the parameter `int totalApplesNeed` could be different.

There is not a strong relation between how many apples the player possesses and how much progress the player has made towards the objective. The objective progress is related to the number of apples the player puts into the bowl. The player can get as many apples as he/she wants to, as long as he/she can put enough into the bowl. For example, the goal of the objective `int totalObjective` is 20, but the player can get more than 20 apples if he/she wants more.

At the moment of putting the obtained apples into the bowl, the game checks whether the player has enough apples for different bowl states by comparing `int totalNumApples` with four limit numbers for the four states. The comparing process is shown as the flowchart in figure 38.

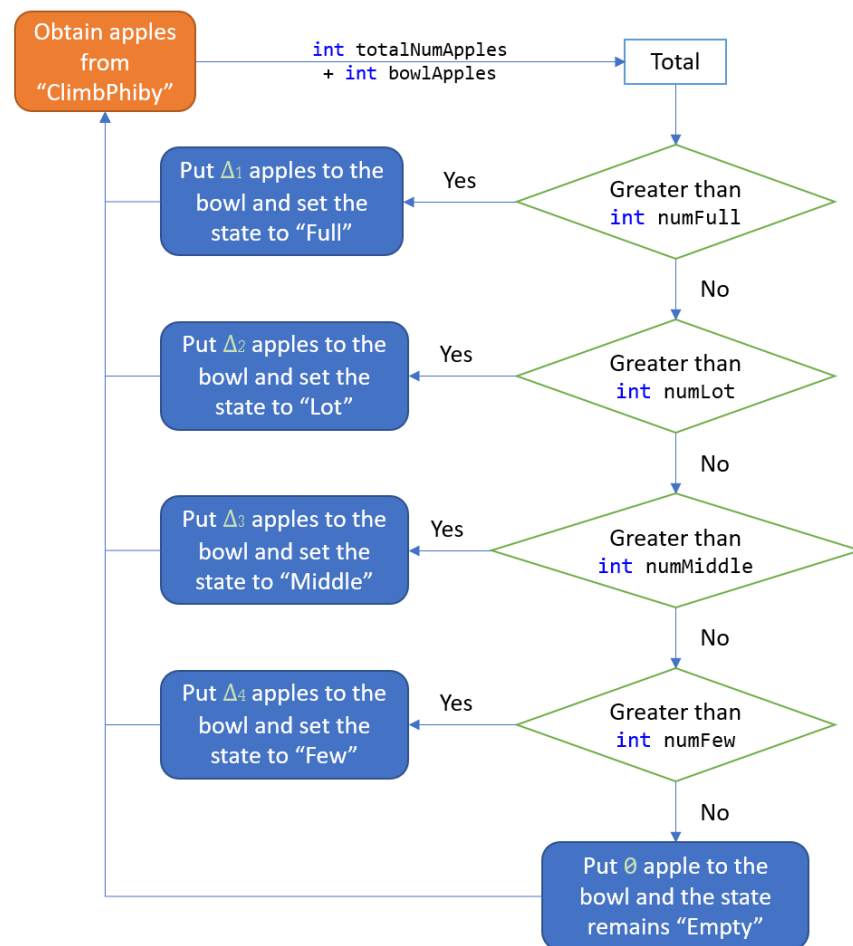


Figure 38. Logic applied when interacting with the bowl

According to this logic, the parameter `int bowlApples` can only be equal to the four limits of the four states. The game always compares the number of apples the player has with the greater value of the four limits, and only continues when the requirement does not meet. The delta values Δ_x present the difference between limit numbers of the new state and the current state. They are calculated with the following formulas.

```
 $\Delta_1 = \text{int numFull} - \text{int bowlApples}$   
 $\Delta_2 = \text{int numLot} - \text{int bowlApples}$   
 $\Delta_3 = \text{int numMiddle} - \text{int bowlApples}$   
 $\Delta_4 = \text{int numFew}$ 
```

The last formula applies only when an empty bowl is filled with `numFew` apples, so the formula is different from the others.

After putting apples into the bowl, the parameters have some new values: `int totalNumApples` is decreased because the apples put into the bowl are removed from the “Tools’ Panel”; `int objectiveDone` shares the same value with the `int bowlApples`, which is increased accordingly to the new state limit number; `int totalApplesNeed` is also decreased, because less apples are needed. The last parameter `int totalApplesNeed` will decrease to 0, when the objective is complete. But `int totalNumApples` is different, because apples are infinite and can always be obtained in “ClimbPhiby” if one wants to.

When the bowl state is “Full”, the objective is complete, and it is the condition to get a “yes” from the question “Enough apples?” in the figure 36 mentioned before. The segment ends when the objective is complete, and the player interacts again with Granny. Granny takes back the key and tells Phiby her worries about the broken bridge, and the checkpoint changes to 3. As the key is no longer in the “Tools’ Panel”, the player cannot enter the hut anymore. The story from then on enters the segment 4, the first segment of the bridge part. Before moves on to the bridge part, it is necessary to mention the mini game scene “ClimbPhiby” and the corresponding mini game simulator view.

4.8.3. Mini game: Climbing apple trees

Although the objective of this project is the creation of the story that takes place in the main island scene, the mini game scenes are an important part of it, so they will be also outlined here. The climbing mini game has existed already in the first version of Phiby’s adventures and the first implementation in Unity was made by César Luaces Vela [12]. Afterwards, different students have modified the scene, the version finally used in this project was created in parallel to this project by Fernando Díez Muñoz [23].

In this current version, there are five apple trees in the main island scene, which can be activated to be “climbed” or let passive. In each game session, other three are activated, which is shown by fallen apples around the trunks of the trees. The more apples, the “higher” the

tree and the more apples can be obtained. A comparison between an active tree and an inactive tree is shown in figure 39.



Figure 39. An active big tree (left), an active small tree (middle), and an inactive tree (right)

In the “ClimbPhiby” scene, the climbing starts with a few meters without apples, which corresponds to the trunk of the tree. After reaching a certain height, apples start to appear and Phiby obtains them (randomly one or two in each movement) when the movements are made correctly (rising the arms in alternating movements). Every movement, a correct or an incorrect one, counts towards the total movements made. This is visualized by the energy bar, where every movement the player does, reduces a small amount of energy. If the total number of movements exceeds the maximum number of movements allowed, the energy drops down to the minimum energy (10%), and the player is forced to leave the game, no matter how many meters he/she has climbed up. As the maximum number of movements allowed ($Max_movement$) is different for each player (fixed by the therapist via web), the energy rested in each movement ($Energy_cost$) is calculated as:

$$Energy_cost = 90\% / Max_movement,$$

Where 90% is the difference between the maximum energy (100%) and the minimum energy (10%).

For example, if 20 movements are allowed, the energy cost should be $90\% / 20 = 4.5\%$. Figure 40 shows an example with 20 movements allowed, where the player has already done 14 of them, shown at the right side of the screen. ($4.5\% * 14 = 63\%$) is the energy cost in that moment, and the player has 37% energy left.



Figure 40. Energy cost for 14 movements in an example with 20 movements allowed

There are two reasons that cause the player to leave the “ClimbPhiby” scene, either the player has climbed to the top of the tree, or the player has run out of energy.

If the game is in keyboard mode, when the player interacts with the trees, the mini games simulator scene is loaded instead. Same as the mini game for the cage, this time the simulator shows the descriptive text about the real mini game “ClimbPhiby”. But different from the cage mini game, this time it can be used to simulate obtaining apples by pressing the space key. At the meantime, energy decreases the same way as in the real mini game scene “ClimbPhiby”. Figure 41 shows the simulator with the description of the mini game before the player pressed any key.



Figure 41. Mini games simulator scene for climbing mini games at checkpoint 2

Under the same condition, with the 20 movements allowed as maximum, hitting 14 times the space key, the energy decreases to 37%, as shown in figure 42. No other hotkeys are set to simulate correct movements and incorrect movements, each time the player presses the space key, an apple is obtained. Figure 43 shows that the energy is at 10% when 20 apples are obtained.

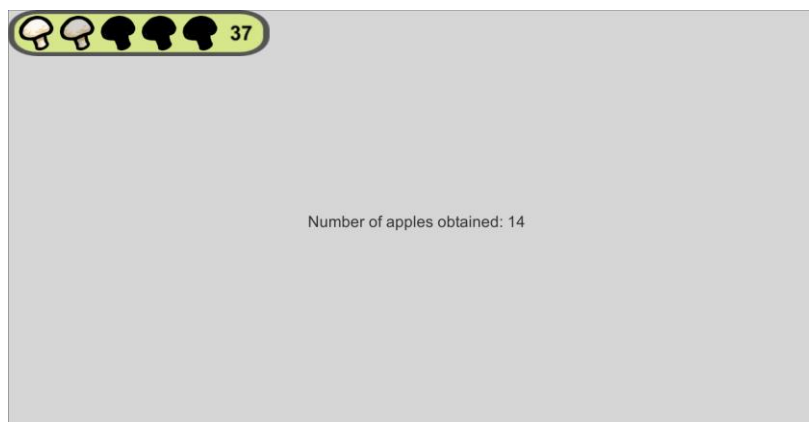


Figure 42. Energy cost for 14 apples in the simulator with 20 movements allowed

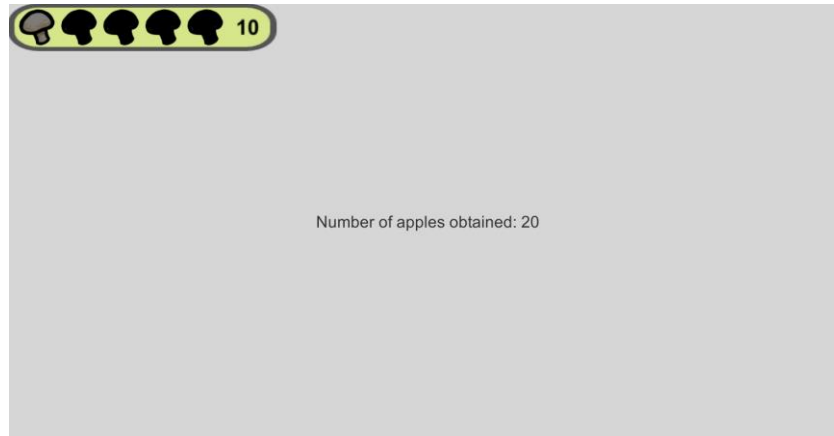


Figure 43. Energy reaches the minimum when 20 apples are obtained, in the simulator with 20 movements allowed

When the remaining energy does not allow more movements, hitting the space key will load the main island scene. While loading, the simulator will give a tip to the player to pick up mushrooms for regaining energy, as shown in figure 44.

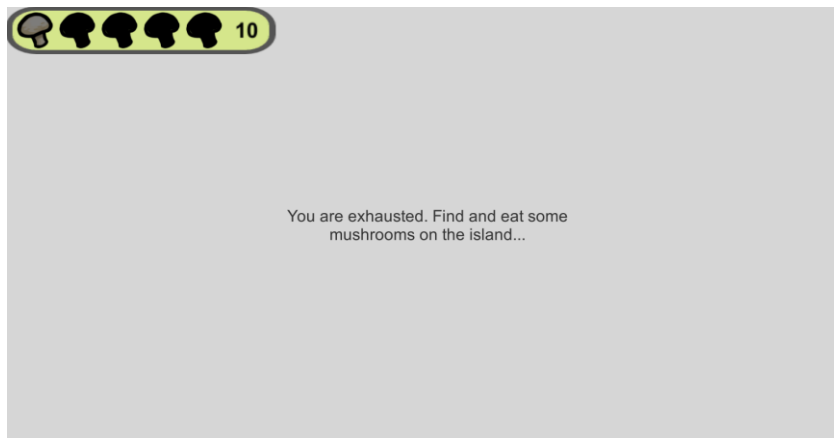


Figure 44. A tip for the player to regain energy

With all these features of the mini games simulator, the player can test the game by obtaining apples and losing energy in keyboard mode as well. For therapists, this can be very useful to set up parameters and having more idea about the energy system by testing the waiting time.

In both modes, when energy is not enough for doing at least four movements, the player does not have access to the mini game, neither the real mini game “ClimbPhiby” nor the simulator. Instead, a tip for the player will pop out as shown in figure 45, to ask the player to rest. The minimum energy for entering the mini game or the simulator can be calculated by:

$$10\% + \text{Energy_cost} * 4.$$



Figure 45. Phiby is not allowed to enter “ClimbPhiby” or the simulator with 10% of energy

4.9. Part three: “The Bridge”

This part contains the other four segments of the first chapter of the game: segment 4, 5, 6 and 7. The game object that marks this part is a bridge, so it is named the bridge part. There are two bridges in the whole island scene, but only “Bridge1” is connected to the side of the river where the chapter one of the game takes place. The other one “Bridge2” is not accessible in this chapter and the term “bridge” used here refers to the game object “Bridge1”.

4.9.1. Layout

It is supposed that Phiby cannot swim, so the bridge is the only way to leave this part of the island and to reach the other side of the river. The bridge was on the island before this project started, but in this project, it has gone through a huge change, first of all, it has been relocated. Figure 46 shows two screenshots taken before and the modification. The game object “Bridge1” is selected in the hierarchy window so the bridge is marked in the screenshots. It is noticeable that the bridge has been moved “up”. This is a decision made after taking into consideration the distance the player has to travel during the game, which is related to the next to be talked about.



Figure 46. The new location (left) and the original location (right) of the bridge

A new NPC rocky is created for the part three of the first chapter. He is an elder brother of Phiby’s who has super strength as his special ability. His name came from the word “rock”, which implies that he is strong and tough like a rock. Because of the special power and the name, he has been located near to the mountain between two huge rocks which is a trap for him, as shown in figure 47.



Figure 47. Location of Rocky and the trap of rocks related to the bridge and the hut

To avoid long distance journey, the bridge is set between the trap and the hut, as the part three starts at checkpoint 3 in front of the hut and the pile of trunks are used for the mini game of this part. To not cause unnecessary searching, the same road from the cage to the hut is extended to the bridge. The bridge is broken at first but repaired step by step during the part three. During the process, the bridge is presented as shown in figure 48.

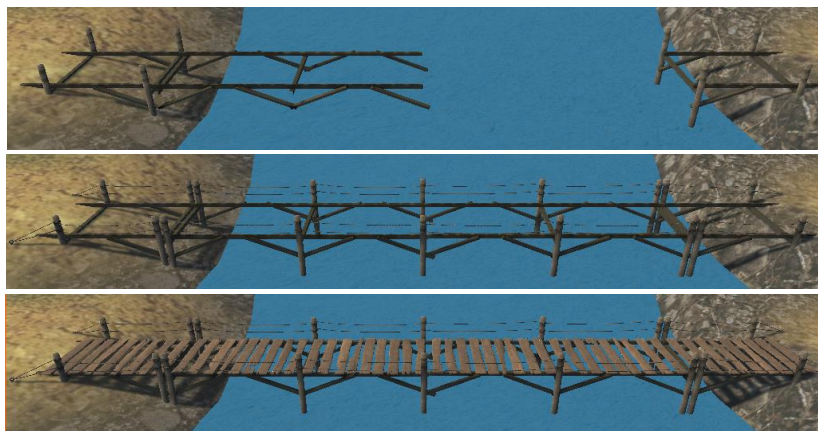


Figure 48. The bridge is repaired during game process

The bridge at first is presented with the broken structure as in the first screenshot in figure 48, and when after the rope part is repaired, it looks like the screenshot in the middle of figure 48. The last screenshot in figure 48 shows how the bridge looks after repairing the planks. During the process of repairing the planks, they are grouped in four parts, and repaired one by one, as shown in figure 49.

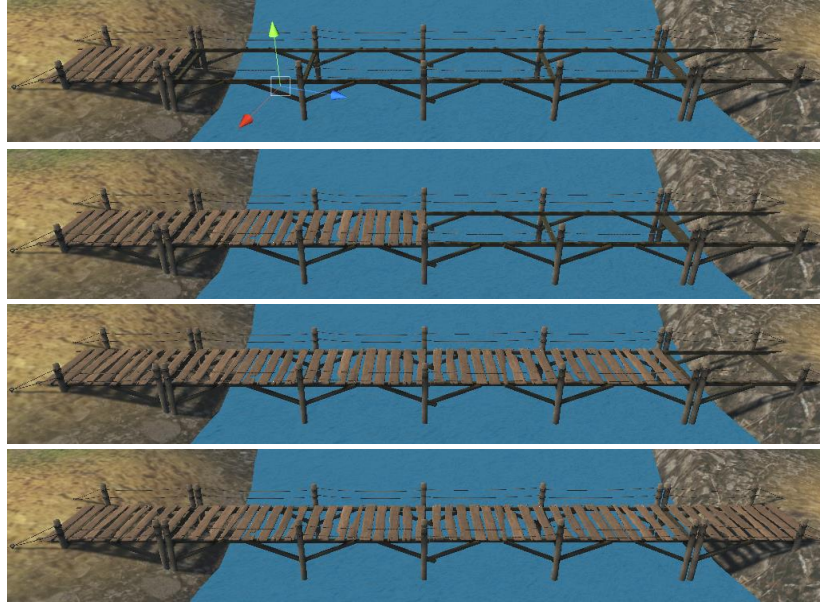


Figure 49. Repairing order for the planks

To achieve this visual effect, the game object used to form the original bridge has been re-grouped. By activating and deactivating the parts one by one, the effect can be easily presented. The original groups of the bridge and the groups after re-grouping them are shown in figure 50.

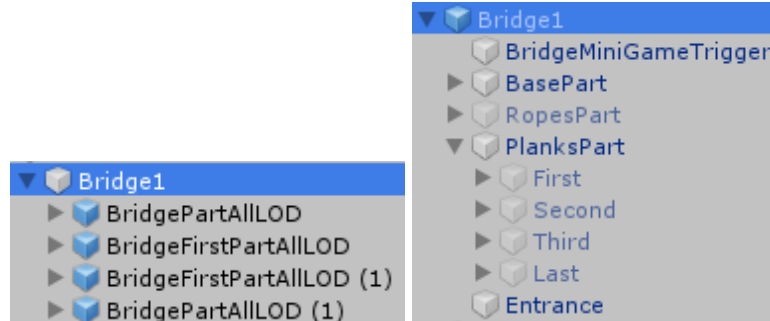


Figure 50. The hierarchy group of the game objects of “Bridge1” before (left) and after (right) being re-grouped

A “BridgeMiniGameTrigger” and an “Entrance” are added to the bridge. Both are transparent game objects that are invisible to the player during the game. The trigger is a semi sphere that detects if the player is in the range. If the player is in the range, different events happen according to the current checkpoint. The entrance is a panel that is used to prevent the player from crossing the unrepaired bridge and falling into the water. It is activated until the bridge is fully repaired. The positions of them are shown in figure 51.

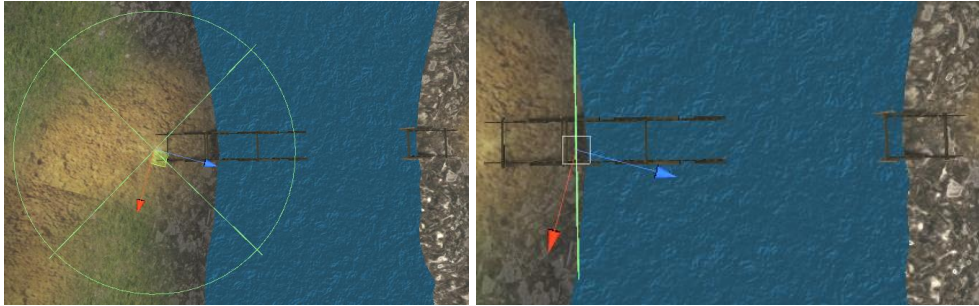


Figure 51. The trigger (left) and the entrance (right) of the bridge

Another important place is where Rocky is trapped at the beginning. He can be found at first near to the mountain supporting two huge rocks, as shown in figure 52. Another trigger is placed to interact with the player, as shown in figure 53.

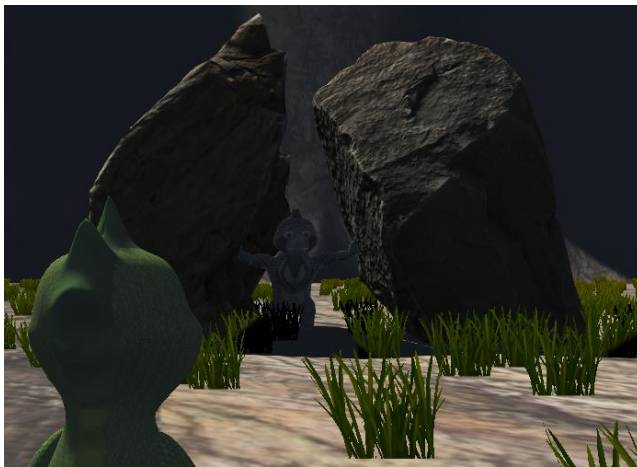


Figure 52. Rocky trapped by two huge rocks



Figure 53. The transparent semi sphere trigger attached to the rocks

After helping Rocky get rid of the rocks, Rocky follows the player using the Unity Nav Mesh Agent. It makes Rocky keep path finding to the game object tagged as “Player” when it is active. Figure 54 shows the configuration of Nav Mesh Agent in the Unity inspector window and the tag of the game object “PhibyKeyboard” as “Player”.

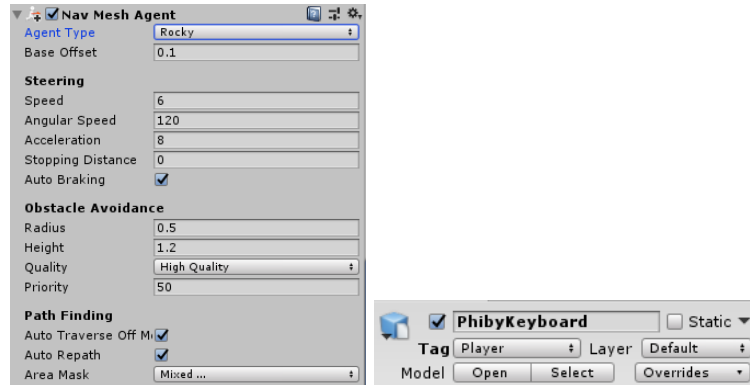


Figure 54. Configuration of the Nav Mesh Agent for Rocky (left) and the “Player” tag for “PhibyKeyboard” (right)

As mentioned in the part two, the pile of wood trunks is relocated. Figure 55 shows the prefab of the game object “Logs”, which is a pile of wood trunks. It is relocated so that the player does not have to go pass the hut to enter the mini game “ChopPhiby” to obtain logs, since interacting with it can trigger the mini game scene in certain checkpoints.



Figure 55. Prefab of the game object “Logs”

4.9.2. Logic

The moment when Phiby gives the key back to Granny, the checkpoint value changes to 3, and a new part begins with segment 4. Comparing to other segments, the segments in this part are short but very important for the story. There have to be four different segments for triggering the different events with different checkpoint values. It is too short to present it in an individual flowchart, so in figure 56, a flowchart for the entire part three is shown. The four green hexagons with a number inside means the checkpoint value changes to that number at that point. The scrolls with numbers are processes to be explained. When explaining, the scroll icons will be used to indicate each process.

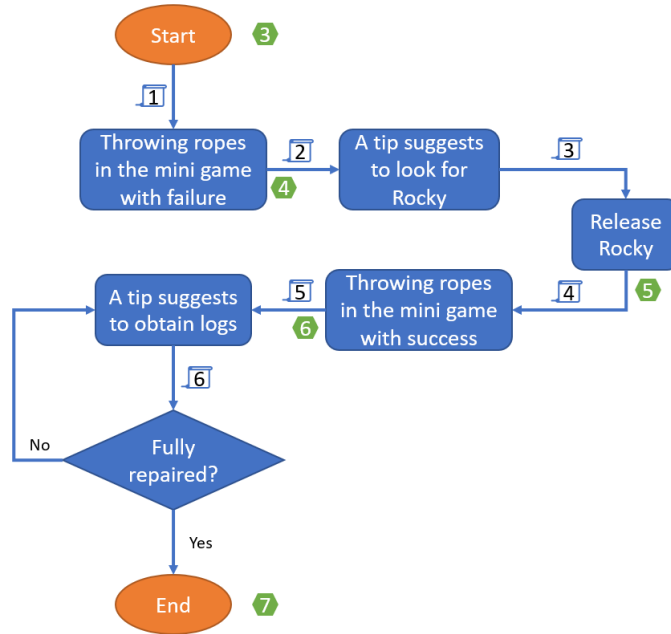


Figure 56. Descriptive flowchart for the part three

1 Leaving the hut, Phiby follows the road and finds the broken bridge. By interacting with the trigger of the bridge, the mini game of throwing the ropes is triggered. As the mini game is not implemented yet, the mini games simulator works instead. This time, the mini game is designed to get a failure result, no matter if the movement is correct or incorrect, and after the mini game, the checkpoint value changed to 4.

2 After the mini game, if the player interacts with the bridge trigger again, a tip pops out telling Phiby to look for Rocky since Rocky has super strength which may help him. This process is totally optional, instead of interacting with the bridge once again, the player can start looking for Rocky directly.

3 Rocky's location is explained earlier. This process is to find Rocky at that location. The trigger attached to the two huge rocks covers a quite large area. It is used to guide the player a little when the player is already near to the destination. If the player enters the area, a tip shown in figure 57 is given with a voice of Rocky telling Phiby his situation (being trapped). After that, if the player goes to the wrong direction and leaves the area, a tip pops up to beg Phiby not to leave, and another tip when Phiby comes back into the area, as shown in figure 58. Finally, the player will find Rocky.

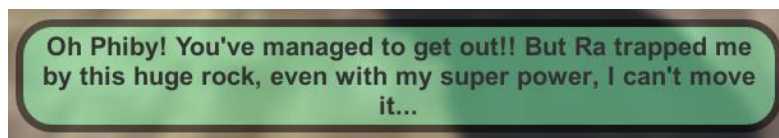


Figure 57. Popped out tip when Phiby enters the area for the first time

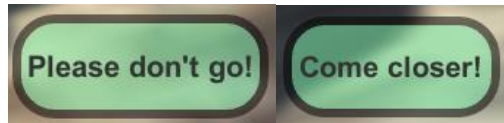


Figure 58. Popped out tips when Phiby leaves the area without releasing Rocky (left) and when Phiby comes back (right)

Continuing with the process, Phiby can release Rocky by interacting with the rocks two times. The first interaction gives the player the reason why Rocky can be released. In the story, Phiby's special ability is to copy the ability of his siblings when they are near to him. So here, as Phiby is near to Rocky, Phiby has super strength. Together with Rocky's super strength, the rocks can be safely moved, and Rocky can get out. The second interaction causes the rocks to be moved to a side visually, the checkpoint value changes to 5.

4 In segment 6, which means that the checkpoint value equals to 5, Rocky follows Phiby. When Phiby is walking or running, Rocky follows at his pace. When Phiby stops, Rocky comes and stands at his right side, as figure 59 shows. Rocky follows Phiby to the bridge, and with the same trigger, the same mini game as in the checkpoint 3 is loaded. But this time, the player is allowed to complete the mini game successfully. After the mini game, the checkpoint value turns into 6.



Figure 59. Rocky and Phiby

5 Checkpoint 6 is the last checkpoint in chapter one. Rocky stops following Phiby and stays near to the bridge waiting for Phiby to come back. When the checkpoint value is 6, interacting with the trigger of the bridge, it creates a new objective for the player, which is to obtain logs by chopping trunks. The parameters `int totalObjective` and `int objectiveDone` mean the number of logs the player has to bring to the bridge to repair the planks and the

number of logs he/she has brought to repair already, respectively. Figure 60 shows an example of the objective indicator for this segment, in which case the player has to collect 6 logs to repair the whole bridge. It is also the default goal of the objective, when there is no personal configuration downloaded from the web.

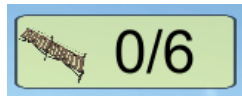


Figure 60. The objective indicator for the bridge

6 This process can be repeated if the player chooses to go to the bridge and put the planks before completing the objective. In this case, the bridge can be repaired at a maximum of four times, since the planks are divided into four parts, as mentioned before. Figure 61 indicates how many planks there are in each part.

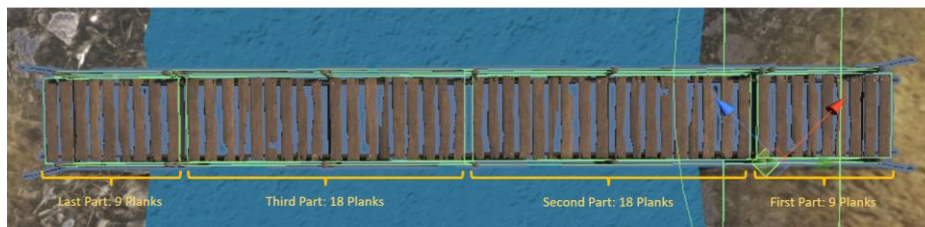


Figure 61. Planks of the bridge being divided into four parts

Since the game objects are fixed and a single plank cannot be separated from the others if they together form a game object, these four parts are defined as: 2 small parts and 2 big parts. The planks are placed in order. Figure 62 is a flowchart that describes the calculation of how many parts to repair when the player interacts with the bridge trigger with logs in the “Tools’ Panel”.

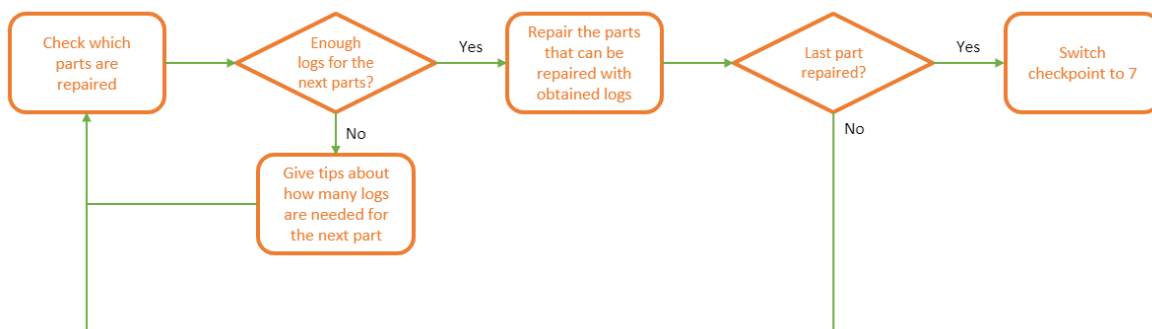


Figure 62. The logic for repairing the planks

When the bridge is fully repaired, the checkpoint value switches to 7 and the game object “Entrance” is deactivated, which means the player can cross the bridge and start the chapter two. Chapter one ends here.

4.9.3. Mini games

As mentioned before, there are two mini games designed to be in this part. One of them has not been created or implemented. There are only theoretic ideas, and it will be mentioned as the bridging mini game. The other one already existed before this project; it is the chopping mini game in a separate scene “ChopPhiby”.

Bridging

This is a mini game to throw the ropes and repair the rope part of the bridge. The mini game is designed to be loaded two times during the chapter one. The first time is in segment 4, the result is designed to be failure. The second time is in segment 6, the result depends on the player, but the player must succeed to go to the next segment.

The core movement for this mini game is to move one arm from back to front. To this moment, no real mini game is implemented. But the simulator acts differently for these two times that the mini game is loaded.

The descriptive text in the simulator is the same for both segments, as shown in figure 63.

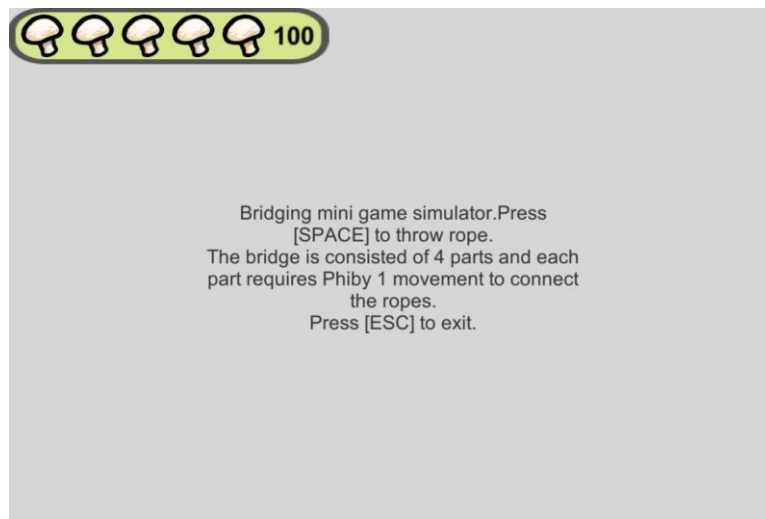


Figure 63. Bridging mini game simulator information

In the information, the number of movements each part requires, “1” in this example, is a parameter passed from the variable `int` `eachPartRopeThrowingTime` in “gameSettingSustain.cs” and it can be adjusted.

But when trying to throw ropes by pressing the space key, for segment 4, it shows the failure result, as shown in figure 64, and for segment 6, it shows the progression, as shown in figure 65. Both, the screen of failure result for segment 4 and the screen of success for segment 6, are followed by the loading of the main island scene.



Figure 64. Results of the mini game bridging in segment 4

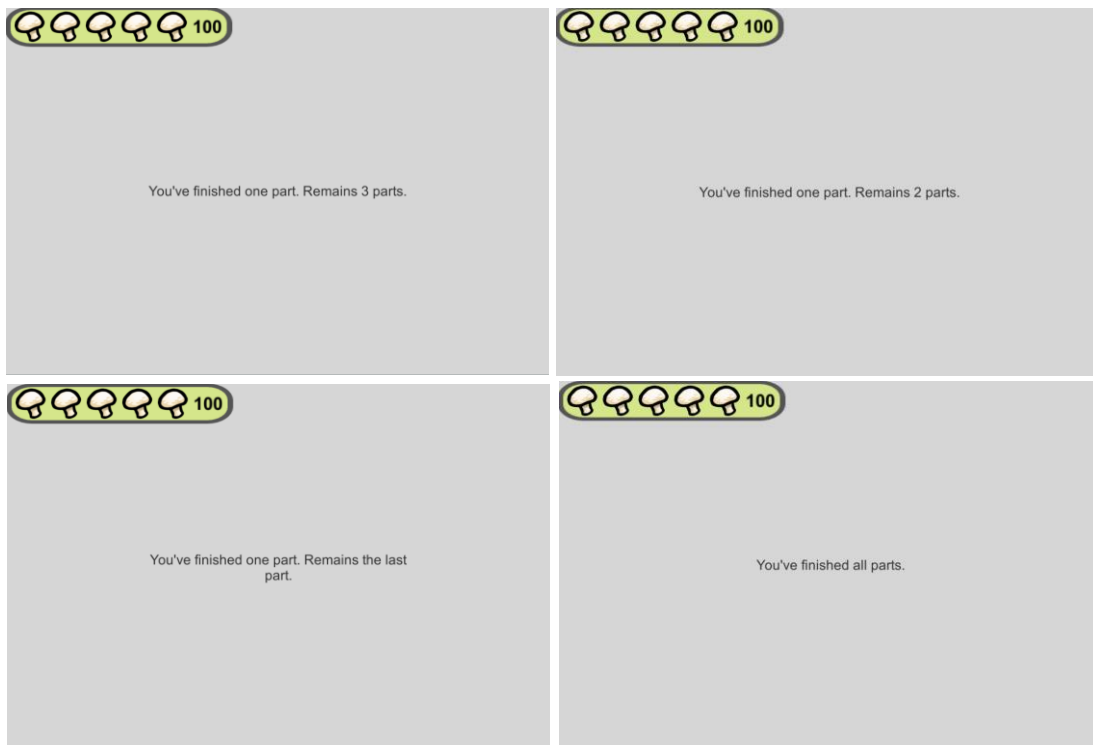


Figure 65. Information of the progression and the final succeed screen

Since the mini game is not developed, the simulator works in Kinect mode as well. Figure 66 shows the simulator information when loading it in Kinect mode. In both, segment 4 and segment 6, the information is the same.



Figure 66. The simulator for the bridging mini game in Kinect mode

Chopping

The planks of the bridge are divided into four parts and as the second and the third part are larger, they count as two units while the other two parts count as one unit each. So, in total there are six units. Each unit needs a certain number of logs to repair. This number is defined in “gameSettingSustain.cs” with the variable `int logsForFirstPart`. Without configuration, by default this number is 1, so 6 logs are needed to complete segment 7. The player obtains one log with each movement (raising and lowering the arm) in this chopping mini game.

The mini game scene “ChopPhiby” was originally created for Unity by César Luaces Vela [12]. Same as the bridging mini game, energy losing during “ChopPhiby” has not been designed, and the scene itself is modified to adapt to this new version but has not been completed after modifying. The player can enter the scene, but no logs will be gained. Figure 67 shows the incomplete scene.



Figure 67. “ChopPhiby” scene

While on the other hand, the simulator is used in keyboard mode, and logs can be obtained. Figure 68 shows the description in the simulator for this mini game.



Figure 68. The simulator for the chopping mini game

Since there is not a design for energy losing during the bridging mini game and the chopping mini game, energy will not decrease neither in keyboard mode nor in Kinect mode for both mini games. But as a prefab, the energy bar can be added and adapted to any scene if it is needed.

4.10. Main menu

After designing the whole chapter one of the story and implementing the design, a main menu is designed to give a better gaming experience. There was an older version menu created by Laura Molero Salazar [10], which has been extended and improved in this project. But as the menu scene was very simple and not offering many functions, it was preferred to create a new scene “MainMenu”.

4.10.1. Re-design

The older version menu scene “MenuScene” was as shown in figure 69.



Figure 69. Menu scene in the previous version

There are three buttons: “Ayuda”, “Jugar”, and “Salir”, which means help, play and quit, respectively. The logo and the buttons were not in a suitable size and the “Ayuda” button was not designed to do anything.

Based on the idea of the older version and how it should be improved, the new main menu scene “MainMenu” is designed. It is much more complicated than the previous version and cannot be shown entirely in only one single figure. By clicking on some buttons, another menu with different buttons appear. For the purpose of avoiding confusion, the first menu shown without the need to click any button will be mentioned as the first level menu, and the menus which can be accessed from the first level menu will be mentioned as the second level menus, and the rest can be denominated in the same manner. To give a rough idea, the first level menu is shown in figure 70.



Figure 70. First level of the main menu

The similar color to the older version is used for this new version, but the background picture has been changed. It is a screenshot of Phiby in front of Granny's hut, showing a little mushroom on the ground with some apple trees in the view. The meaning of the background picture is to show some important new features of the game. It can be changed easily so the idea is to change the background picture to a new screenshot that shows roughly the new features every time when there is a big update.

Buttons are designed to be half transparent. When the graphic pointer of the mouse moves to a button, the button turns into a lighter shade and is no longer transparent, as shown in figure 71. And at the moment the player clicks a button with the pointer, it becomes darker in shade and not transparent as shown in figure 72.

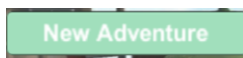


Figure 71. Color of a button when the pointer stops on it



Figure 72. Color of a button the moment when it is clicked

The first difference between the two screenshots of the older version and the new main menu should be the game title "Phiby's Adventures". The game title in the new main menu is not shown with the first level menu but before the first level menu even being shown, and for convenience, it is mentioned as the welcome screen. The welcome screen, shown as in figure 73, shows only the game title in the upper middle of the whole screen. The logo of the game title is re-designed using Paint 3D. Comparing figure 73 with figure 70, it is noticeable that the first level menu is on a partially transparent panel.



Figure 73. Welcome screen with the game title

A simple visual effect is added that the logo gradually appears and disappears, before the first level menu is shown. If the player goes back to the main menu scene from the main island scene, this visual effect does not show, and this is controlled via a Boolean value `bool firstLoad` in “gameSettingSustain.cs”.

The effect is done with a timer using `IEnumerator` to change the transparency of the logo, similar to the effect of mushroom icons glittering in the mini map. The transparency value starts to increase from 0, which makes the logo totally transparent at the beginning. When the transparency value increases to the maximum, it stays unchanged for 2.5 seconds before starting to decrease and finally reaches the minimum 0. Figure 74 shows a general idea of the visual effect.



Figure 74. Starting effect of the main menu scene

A music is added as background music. The music piece is “Mario’s way” made by Gianni Caratelli, who published it on the website Freesound [24]. The music piece is about 47 seconds but is set to loop in the main menu scene, so the music never stops.

Menus of other levels are of the same style. Except for the first level menu, all menus of other levels have a “Back” button, which upon clicking the screen goes back to the previous menu. As in this project, English is the visual language, the whole main menu is also designed in English. All buttons in the whole menu scene have their own function, which will be explained later, so menus of other levels will be shown when associated buttons are being mentioned.

4.10.2. General functionality

There is only one script for the main menu scene, “MainMenu.cs”. All necessary methods are programmed inside this script. Buttons’ on-click events are set via the Unity inspector window, some of them call methods in “MainMenu.cs”, some of them activate or deactivate game objects.

The four buttons in the first level menu have similar functions with the ones of the older version. Instead of having only one button to load the main island scene, there are two buttons: “New Adventure” and “Continue Adventure”. Comparing to the older version, the player has one more option not to play the game from the beginning but to continue from the last checkpoint he/she has reached by clicking “Continue Adventure”. This is based on the new feature of game data saved after quitting the game. The mechanics and logic behind this feature will be further discussed in 4.11.

The “Quit” button is used to quit the game when the game is built into an application and runs outside of Unity. It does not work when running the game in Unity, since the method `Application.Quit()` is only for built applications.

The “Setting” button leads to the second level menu. To avoid a too complex first level menu, all other possible functions are put into “Setting”. By clicking on “Setting”, the second level menu appears. In the Unity inspector window of the buttons which leads to the next level menus, the on-click event is set to deactivate the current level menu and activate the target menu. In this case, it deactivates the first level menu and activates the second level menu. Figure 75 shows the chosen on-click event in the Unity inspector window for “Setting” button.

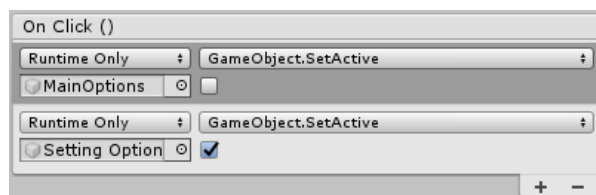


Figure 75. On Click() method setting for the button “Setting”

The second level menu only have two buttons, and one of them is the “Back” button for going back to the first level menu. Figure 76 is a screenshot of the second level menu. All “Back” buttons have similar on-click events. Upon clicking, the current menu is deactivated, and the previous menu is activated. The configuration is shown in figure 77.



Figure 76. Second level menu

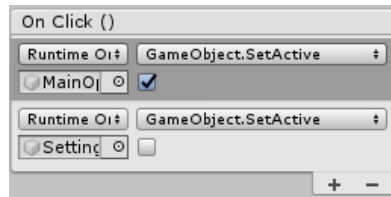


Figure 77. On Click() method setting for the button “Back” in the second level menu

Another button “Editors’ tool” is designed especially for developers. By clicking on it, the third level menu shows up. The functions offered in the third level menu is presented in the next part, as they are quite unique.

4.10.3.Editors’ tool

Figure 78 shows the third level menu known as “Editors’ tool”. As its name implies, it offers special functions to meet the needs of editors and developers.

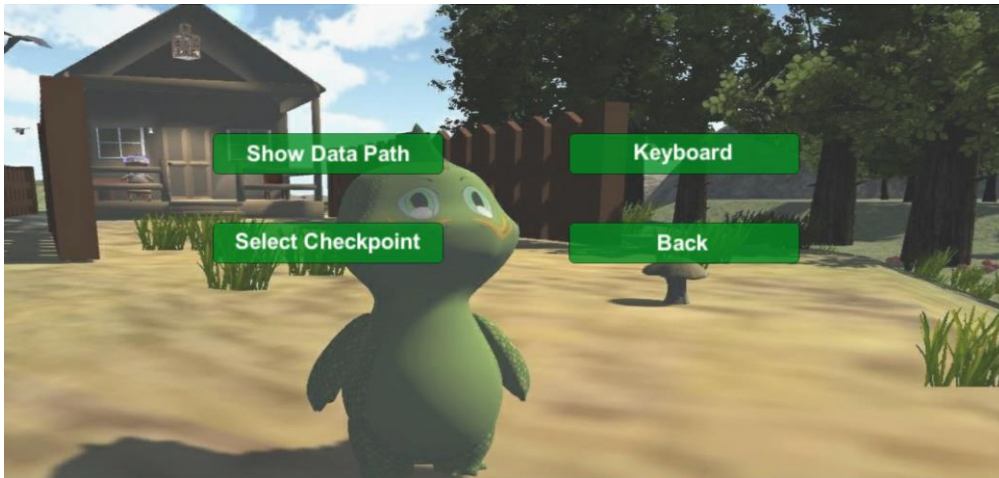


Figure 78. Third level menu “Editors’ Tool”

The first button “Show Data Path” is used to show the directory path where the game data files are saved. By clicking on it, the path is shown with a button which can copy the directory to clipboard as shown in figure 79. This is useful because the path can be different in different computers. The text on the button “Show Data Path” changes to the “Hide Data Path”, and by clicking it again, it returns to be the original text and the path and the “Copy” button are hidden.

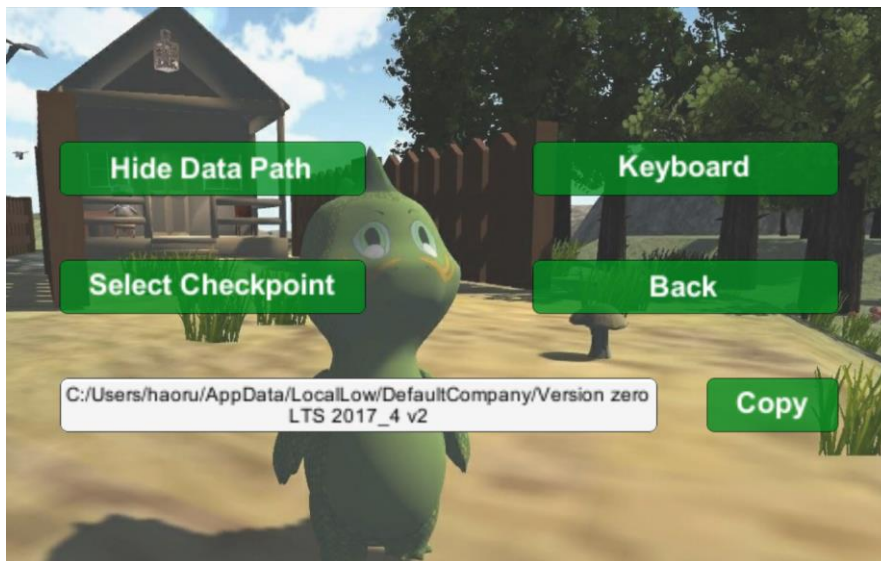


Figure 79. Showing directory after clicking the button “Show Data Path”

The second button shows “Keyboard” or “Kinect” depending on which control mode is selected. At the moment of writing, in the “gameSettingSustain.cs”, the default control mode is set to be keyboard mode. Therefore, the button shows “Keyboard” without clicking. But upon clicking, the Boolean variable `bool isKinect` is set to true, and the text on the button is

changed to “Kinect”. The control mode in the main island scene will be identical to what is shown on the button. The two forms of the button are shown in figure 80.



Figure 80. Two visual forms of the button for switching control modes

The third button “Select Checkpoint” leads to the next level menu, which is shown in figure 81. In this fourth level menu, the editor or the developer can select a checkpoint where they want to enter the game, so they do not have to pass through all for testing or bother to change it in the script repeatedly. Without clicking, all checkpoints are folded inside the two visible buttons “Chapter 1” and “Chapter 2”. By clicking on it, options are unfolded as shown in figure 82.



Figure 81. Fourth level menu with checkpoint options folded



Figure 82. Fourth level menu with checkpoint options unfolded

Checkpoint options are presented in circles with the corresponding number. Even though the “Chapter 2” button is put into the menu already, since the chapter 1 is the only finished

chapter, the checkpoint 7 is undefined. The “Chapter 2” button is there for a design purpose and to give a guidance to future developers where to put future checkpoint options.

All checkpoint selecting buttons work simply by replacing the checkpoint value `int` checkpoint accordingly in “gameSettingSustain.cs”, and then run the main island scene. There is no connection with other checkpoints, in other words, the game starts at the selected checkpoint without having passed through any previous checkpoints, so some variables have no values, but in this case, the menu asks to select them. For example, it has been mentioned that in segment 2 the goal of the objective for the bowl is set a brief moment before entering the segment 3. That is said, when the checkpoint is 2, there must be a goal of the objective for the player to achieve.

On normal gameplay, the goal of the objective is impossible to be absent: if the player continues playing without quitting in segment 2, the goal of the objective does not interfere with the following gameplay; if the player chooses to quit during the segment 3, upon saving the game data, as the checkpoint value is 2, the goal of the objective is saved for the next time. No problem will be caused for either of the possibilities. But if an editor or a developer needs to go to the checkpoint 2 without playing the segment 1 or segment 2, the absence of the goal of the objective will cause problem. To solve this problem, a fifth level menu is created and shown as figure 83.



Figure 83. Fifth level menu for setting goal for the bowl objective

There is a descriptive text shown at the top of the screen, telling the editor or the developer what the menu does. In the middle of the screen, there are an input field and two buttons. The input field is set to be an integer input field so that only integer numbers can be entered, avoiding possible errors. At the right side, the button “Random” gives a random integer number from 0 to 20 to the input field upon clicking. The given number can still be manually modified. If the editor or developer is satisfied with the number, given randomly or manually, he/she can click on the “OK” button, and the game will start from checkpoint 2 with the goal

corresponding to the given number. Figure 84 shows the visualization when the given number is 12.



Figure 84. Setting the goal of the objective to 12 for the checkpoint 2

A few functions of the main menu are about the data saving as files. These parts will be further described in the next subchapter.

4.11. Game data saving

Game data are important for most games. Before this project, there was not an efficient game data saving system. Every time the game starts, it was a new game that starts from the very beginning. But Phiby's Adventures v2 as an adventure game will offer the player hours of game experience, so saving game data is an essential step. For this purpose, two types of data saving are developed in this project: data saving between scenes and data saving as files after quitting the game.

4.11.1. Data saving between scenes

The game consists of various scenes and has to keep switching between them. In Unity, there are two ways to load a new scene: single mode and additive mode. Using single mode to load a new scene will cause all current scenes to close, while using additive mode to load a scene will add the new scene to the currently loaded scenes. For not consuming too much memory and improving game performance, single mode is used to load scenes in this project.

But to load a new scene, all current loaded scenes are destroyed including the data of the scenes. These data may be needed in other scenes, for example, apples and trunks the player has gained in "ClimbPhiby" and "ChopPhiby" are also needed in the main island scene.

There are various ways to solve this problem, and in our case, Juan Alberto García Fernández [15] created a script "gameSettingSustain.cs" that uses a Unity method to keep it active during the game. It is attached to a temporary game object "gameSettingObject" that is created using code every time the game runs. At first, "gameSettingObject" is created every time in the awake method `void Awake()`, which is called before the start method `void Start()`, of "GameManager.cs", as it is one of the first scripts to run at the beginning. But later as the game developed, a new main menu scene is created, and the game is supposed to run the main menu scene as the first scene. For being able to create the game object "gameSettingObject" at the very beginning of the game, the method used to create this game object is copied to the main script of the main menu scene "MainMenu.cs". The original codes are not removed from "GameManager.cs" for the convenience of the developers when they want to test the main island scene quickly without entering from the main menu. Although, in a future when the game is complete, these associated codes should be removed to avoid possible problems.

The script "gameSettingSustain.cs" calls in the awake method a Unity method `DontDestroyOnLoad(Object target)`, which in this case, the parameter is `this` so that the game object that the script is attached to "gameSettingObject" is the target object. This method can protect the target object that is passed as a parameter from being destroyed during the game even new scenes are loaded. It creates a scene called "DontDestroyOnLoad" and the game object "gameSettingObject" is created inside of it, as figure 85 shows.

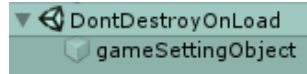


Figure 85. “DontDestroyOnLoad” scene and “gameSettingObject” in the Unity hierarchy window

Same as “GameManager.cs”, “gameSettingSustain.cs” was expanded a lot because it is frequently used to store and read all kinds of important global temporary parameters and data. As the game develops, more and more parameters and data will be saved to the script. Figure 86 shows the existing public parameters of the script in the Unity inspector. Nearly all of them are explained in previous subchapters.

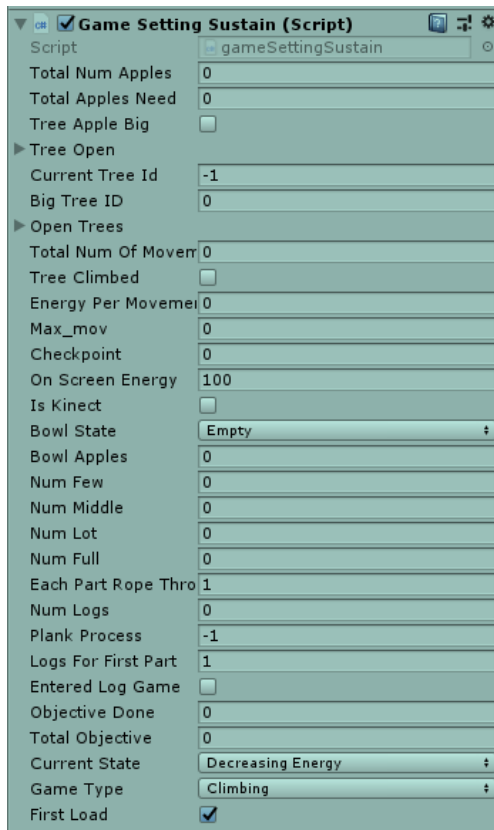


Figure 86. Public parameters can be modified directly in the inspector

All these parameters can be changed manually through the Unity inspector window for testing purpose. Some parameters are assigned during the game process so through the inspector window, developers can also test whether they are assigned correctly or not.

This data saving process is not a choice of the player, so there is no visualization about it. The player does not need to know and will not be notified by any means about which data are passed from one scene to another, since the purpose is to use the obtained data in a different scene and improve the continuity.

4.11.2. Data saving as files

It is not enough to just save data between scenes. The meaning of the new checkpoint system is to save the game progress for the player so that the player can continue at the same point the next time. If the game data cannot be saved in a local file and load the next time the player runs the game, the whole checkpoint system will not work.

There are also different ways to save game data as local files. A simple common way is to save them into a JSON file, but it also makes the local save files easy to read and modify by the player because of the high readability. The game progress should not be easily modified by the player, so it is better to find a way to make it not readable by human. And the solution is to save the data into binary files. Figure 87 shows the different readability of the binary data file generated in this project (cp2.dat) and an example of a random JSON data file (JSON_example.json). With both files opened with Microsoft Notepad, it is clear that the JSON file is much more readable than the binary file.

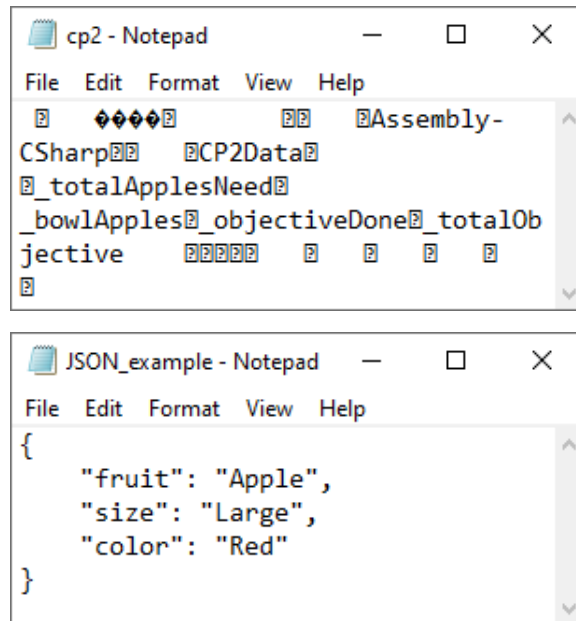


Figure 87. A binary file (up) in comparison with a JSON file (down)

There are basically two parts: data saving and data loading. Since they have been programmed and visualized in different scenes, they need to be discussed separately.

Process of data saving

Data saving should be done every time the player quits the game. To visualize the process, a half transparent panel with a question and options to save game data and load the main menu scene is created as shown figure 88. It can be activated by pressing the Esc key in the main island scene.



Figure 88. Options to save game progress in local files

As seen in the figure, this panel and the main menu scene are designed in a similar visual style. There are one question and three options. By reading the question, the player will understand that he/she must choose if he/she wants to save the game progress. By clicking on “Yes”, new local binary data files are created and saved to a local folder, and then the main menu scene is loaded. By clicking on “No”, the main menu scene is also loaded, but no file is created or saved. This option is designed to offer an option to the player if anything in the last gameplay does not go as he/she wishes, and the player prefers not to save the progress. By clicking on “Cancel”, the main menu scene is not loaded, and deactivate the panel and continue the game. Nevertheless, for the therapist it is very important to receive data about every session played so that he/she can monitor the exercises. That kind of data will be stored on different files and then uploaded to the medical website Blexer-med. This part still has not been implemented in the game and was not objective of this bachelor’s degree project.

The method `public void SaveData()` that creates the binary file and saves it to a local folder is programmed in “GameManager.cs”. It creates an empty file with a file stream “FileStream” and uses a binary formatter “BinaryFormatter” to serialize the data into binary and put the data into the created file. The saving path is defined by the persistent data directory obtained using `Application.persistentDataPath`, which is different in different computers or operating systems. Files that are saved in a persistent data directory will not be deleted or replaced by game updates. Although by now this is not an essential function, it is necessary at the time when designing a method to save game data to be useful in a long term. Doing this also prevents that a manually set directory is not correct in other computers or operating systems. The directory path in windows is usually a path with a standard pattern like this: `%userprofile%\AppData\Local\Packages\<productname>\LocalState`. For example, at the time of writing this thesis, the directory path in the computer used to develop this project is: `C:\Users\haoru\AppData\LocalLow\DefaultCompany\Version zero LTS 2017_4 v2,`

where the product name and “company” can be changed if needed. Figure 89 shows the content in the mentioned directory.

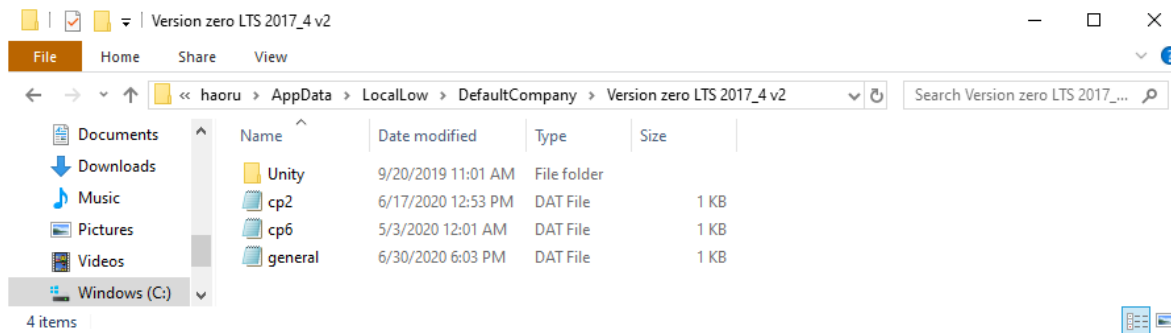


Figure 89. Game data files saving directory and the saved data files

This method does not possess neither any parameter to be passed nor any result to be returned, although it does function differently on different checkpoints. As shown in the previous figure, there are three data files of “DAT File” type in the game data directory. The general data file “general.dat” is generated whenever the player wants to save the game process, and the data for checkpoint 2 “cp2.dat” and the data for checkpoint 6 “cp6.dat” are only generated when the player saves the game process at checkpoint 2 and checkpoint 6, respectively.

Three scripts are created for three types of data: the general data, the checkpoint 2 data, and the checkpoint 6 data. Each of the scripts contains a class that defines which variables in “gameSettingSustain.cs” are there to be saved. They are not attached to any game object and all classes apply the [Serializable] attribute to make them serializable to be encrypted into binary data.

The general data class is defined as the following:

```
GeneralData(int checkpoint, float onScreenEnergy, int totalNumApples, int numLogs)
```

These data consist of the checkpoint value, the energy shown on screen, the number of the basic items (apples and logs) in the “Tools’ Panel” at the moment when the player quits the game. These four values are enough to rebuild the status of the player in most checkpoints, but in checkpoint 2 and checkpoint 6, to protect the player from having to re-do the objective of putting apples in the bowl and repairing bridge from the beginning, extra information needs to be saved for these checkpoints.

The checkpoint 2 data class is defined as the following:

```
CP2Data(int totalApplesNeed, int bowlApples, int objectiveDone, int totalObjective)
```

The checkpoint 6 data class is defined as the following:

```
CP6Data(int plankProcess, bool enteredLogGame, int objectiveDone, int totalObjective)
```

Both classes contain necessary data to set the game back to when the player drops it in these two special checkpoints, with the progress the player has done the last time playing. At the moment of saving game data into files, the method `void SaveData()` detects the current checkpoint value, and creates new files or overwrites existing files according to the checkpoint.

Process of data loading

The option to load game data from files is presented in the main menu as the option to “Continue Adventure”. Unlike the saving method `void SaveData()`, the loading method is programmed in the only script of the main menu scene “MainMenu.cs”.

The process of loading is basically a reversed process of saving. Another file stream “FileStream” is used to open and read the binary data files and another binary formatter “BinaryFormatter” is used to decrypt the read binary data to values for variables and pass them to “gameSettingSustain.cs”.

As for checkpoint 2 and checkpoint 6, there are more than one data file “general.dat”. Different from the saving process, in the main menu scene before loading any data, the value of the variable `int checkpoint` is set to be 0 as default. Therefore, the general data file “general.dat” must always be read first. By reading it, the method passes all the variables saved in the general data file to “gameSettingSustain.cs”. Then, as all other times reading a variable from “gameSettingSustain.cs”, the method reads the value of checkpoint, which is already changed to the value read from the data file “general.dat”, and if the value is equal to 2 or 6, “cp2.dat” and “cp6.dat” will be load, respectively.

In the case of clicking “Continue Adventure” when no file is found in the persistent data directory, a debug log is generated, and the game starts from the beginning with all values in “gameSettingSustain.cs” as default.

With the two processes, game data are managed to be saved, read and load in file form. Also, as the data being encrypted and decrypted during the processes, the player will not be able to modify any values to cheat the game.

5. Conclusion & future work

5.1. Conclusions

After all these descriptions and explanations of the work done in this bachelor's degree project, looking back to the introduction and the mentioned objectives, it is time to make a conclusion.

With a detailed and complete chapter for the story, a solid base is built to achieve the first objective. A clear objective is set for the game as Phiby has the responsibility to stop Ra from destroying the island. Two important new NPCs, Granny and Rocky, are designed and created considering their roles in the game with plenty of dialogues and importance for promoting the development of the story. It has been successfully implemented the checkpoint system as planned, with clear goals for each segment and, step by step, it helps the player to finish the whole chapter. The energy system managed to effectively implement the idea of controlling the amount of exercises the player takes, with the mushrooms all over the designed area. And the game data saving system makes a big difference to the game that now does not need to be played from the beginning to the end in one single time, and the player can enjoy it for a long period of time when the game is completely developed. This new characteristic extremely increased the possible playing time. With these implemented features, the objective one is considered to be achieved.

In this bachelor's degree project, the information for the player is given in various ways. The inventory system implemented as the "Tools' Panel" with a maximum of 6 slots will be enough for most future cases. When designing the dialogue system, it was taken into consideration that it should be easy to understand, both the text and the audio messages. The dialogues are also designed to provide information about the story. Finally, with the game objectives indicator, the player can clearly know how many more apples or logs he/she needs. The objective 2 planned in the introduction chapter is well implemented as well.

The combined model of Phiby (keyboard and Kinect mode) has gone through hundreds of tests and the switching works ideally. Also, the developer mode to select the checkpoint wanted, greatly improves the convenience for the testing process and avoids wasting time. Combining with the newly created menu, developers does not need to modify checkpoint values in the script "gameSettingSustain.cs" between tests anymore, instead, selecting the wanted checkpoint via the menu is all that needs to be done. The objective 3 is successfully achieved.

5.2. Possible future work

From the experience with this project, the best way to continue is developing the game chapter by chapter. A lot of systems and features designed in this project are not only designed for the chapter one, but also for the rest of the future story. With the purpose of not making the game too complicated with too many options and information on the screen, creating much more new systems and features is not recommended. But if someone finds it necessary, it may be a good idea to design a submenu in the main island scene, similar to the saving data screen explained in 4.11.2. It would also be important to implement a system that makes it possible to operate everything in Kinect mode with gestures: open menu, pause the game, check the game progression, etc. as the users are placed in a distance of 1-2 m from the screen.

There was another system planned but there was no time to implement it. The draft plan is attached to this thesis as ANNEX II. It is about an advancement panel and notifications, which means to visualize the checkpoint system. The checkpoint system implemented in this project is not visualized to the player, and it could be more encouraging if the player could actually know their total progression in the game. There are names for each checkpoint as written in the file attached to ANNEX I, which are also designed to be advancements.

There are also many new ideas during the development of this project, but due to limited time and energy, they are still rough ideas.

The first to be mentioned is the visualization of the different abilities. As designed, Phiby copies the abilities of his siblings when they are near to him. The idea is to distinguish visually the abilities Phiby has in each moment. It is not only good for the story designed, but also useful for players to understand the concept of copying abilities.

Speaking of the visualization, the “mini games simulator” and the “Tips Container” are designed with fully function but can be improved if there present the pictures of the different mini games and different NPCs.

By far, only one data file of each type can be saved. If a data file already exists and the game tries to save another file of the same type, the existing file will be replaced. It is likely that with a multiple saves system, the player can have more fun with the game. The player will be allowed to have different save files and could load the one he/she wants to play at different times.

Setting a password for the editors’ tool could also be a necessary in future work before the game is ready to be launched. It is not reasonable that a patient as a player can go to the menu and select the checkpoint to skip the necessary game process. Also, the Kinect mode should be set as the only control mode for a patient.

There is still a long way to go since only the first chapter is finished. For a big project like this, it will need many people to share the work and cooperate. But in all conditions, the

consistency of the story should be highly valued, since a good adventure game is based on a good story.

6. References

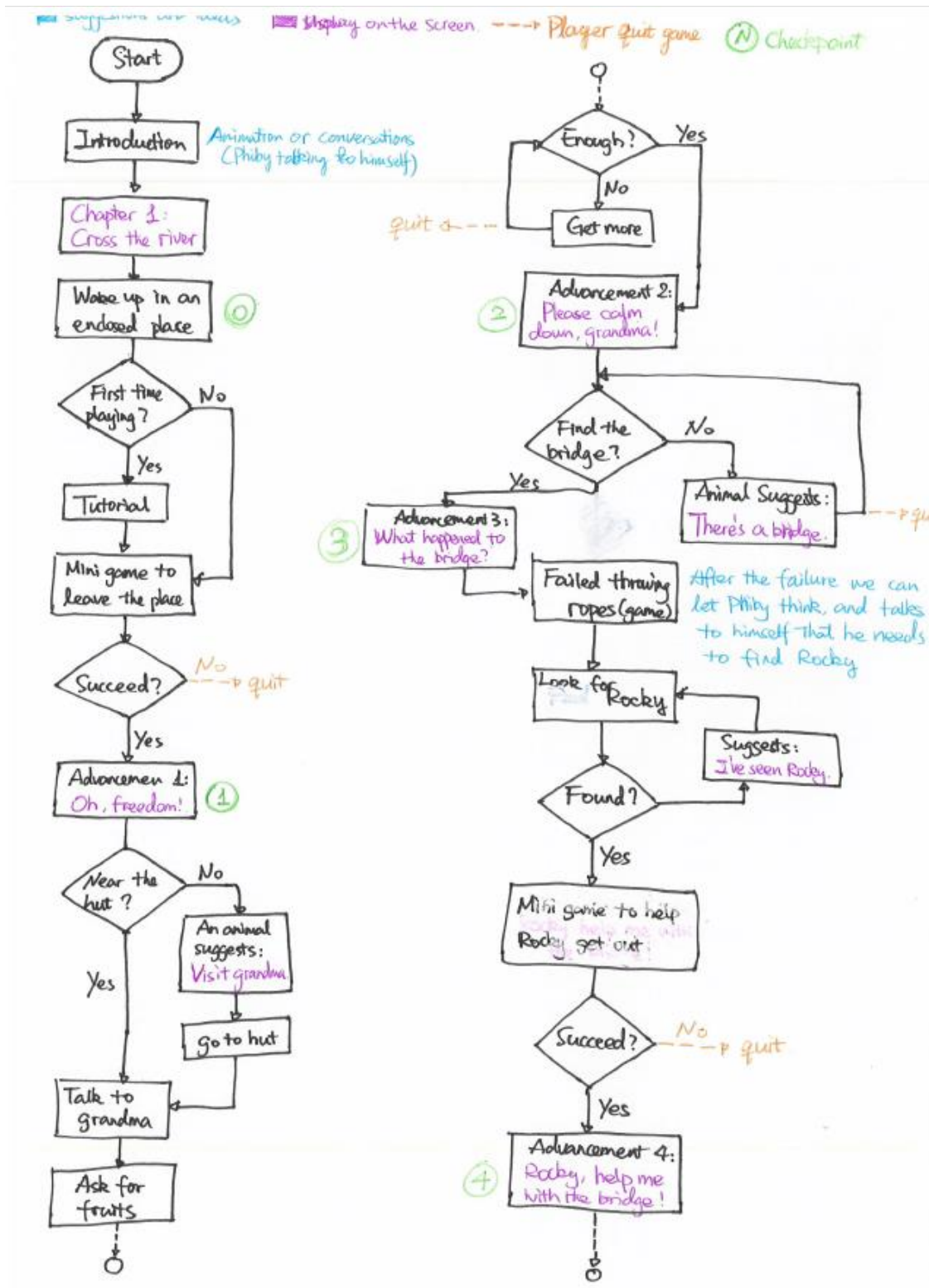
- [1] “Kinect for Windows v2 (Microsoft)”, DepthKit.
<https://docs.depthkit.tv/docs/kinect-for-windows-v2> (Accessed Jul. 17, 2020).
- [2] *Blender*, Blender Foundation. Available: <https://www.blender.org/>
- [3] *Unity3D*, Unity. Available: <https://unity.com/>
- [4] MindMotion. <https://www.mindmotionweb.com/> (Accessed Jul. 17, 2020)
- [5] MIRA. <http://www.mirarehab.com/> (Accessed Jul. 17, 2020).
- [6] WalkinVRDriver. <https://www.walkinvrdriver.com/> (Accessed Jul. 17, 2020)
- [7] PRIME-VR2. <https://prime-vr2.eu/> (Accessed Jul. 17, 2020).
- [8] M. Khademi, L. Dodakian, H. M. Hondori, C. V. Lopes, A. McKenzie, S. C. Cramer. “Free-hand interaction with leap motion controller for stroke rehabilitation”. *UC Irvine*. <https://escholarship.org/uc/item/4p831550> (Accessed Jul. 17, 2020).
- [9] P. M. Kato. “Video Games in Health Care: Closing the Gap.” *Studylib*.
<https://studylib.net/doc/18833632/video-games-in-health-care--closing-the-gap> (Accessed Jul. 17, 2020).
- [10] L. Molero Salazar, “Diseño e implementación del entorno para el juego serio terapéutico “Phiby’s Adventures v2”,” B.S. thesis, Dept. Teoría de la Señal y Comunicaciones, The School of Telecommunications Systems and Engineering, Technical Univ. of Madrid, 2019.
- [11] M. A. Gil Gil, “Integración y calibración de movimientos captados mediante la cámara Kinect para el juego serio terapéutico “Phiby’s Adventures v2”,” B.S. thesis, Dept. Teoría de la Señal y Comunicaciones, The School of Telecommunications Systems and Engineering, Technical Univ. of Madrid, Madrid, 2019.
- [12] C. Luaces Vela, “Diseño e implementación de un entorno virtual de ejercicios físicos, basados en captura de movimiento,” B.S. thesis, Dept. Teoría de la Señal y Comunicaciones, The School of Telecommunications Systems and Engineering, Technical Univ. of Madrid, Madrid, 2019.

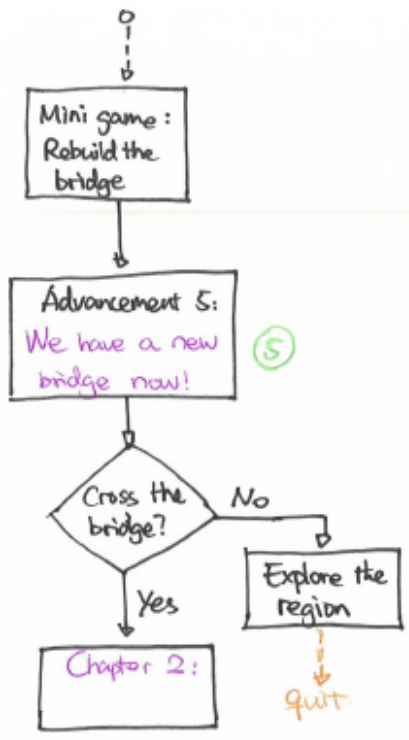
- [13] Sev_4. "FOOD free." Unity Asset Store.
<https://assetstore.unity.com/packages/2d/textures-materials/food/food-free-145841>
 (Accessed Jul. 17, 2020).
- [14] Mikołaj Spychał. "Toadstools Pack - Photoscanned." Unity Asset Store.
<https://assetstore.unity.com/packages/3d/environments/toadstools-pack-photoscanned-70294> (Accessed Jul. 17, 2020).
- [15] J. A. García Fernández, "Diseño e implementación de una lógica de ajuste terapéutico para el juego serio "Phibys Adventures 3D"," B.S. thesis, Dept. Teoría de la Señal, The School of Telecommunications Systems and Engineering, Technical Univ. of Madrid, Madrid, 2020.
- [16] OpenClipart-Vectors. "Tree log." Pixabay.
<https://pixabay.com/ko/vectors/%ED%8A%B8%EB%A6%AC-%EB%A1%9C%EA%B7%B8-%ED%83%80%EB%9D%BD-%ED%95%9C-%EB%A7%8C%ED%99%94-576846/> (Accessed Jul. 17, 2020).
- [17] *PNG Transparency Creator*, Online PNG Tools. Accessed: Jul. 17, 2020. [Online].
<https://onlinepngtools.com/create-transparent-png>
- [18] A. Vázquez Chaves, "Informe de prácticas sobre la implementación de elementos diferentes en el videojuego serio Phibys Adventures v2", The School of Telecommunications Systems and Engineering, Technical Univ. of Madrid, Jun 2020.
- [19] "Checkpoints (Concept)." Giant Bomb.
<https://www.giantbomb.com/checkpoints/3015-292/> (Accessed Jul. 17, 2020)
- [20] Burak Taban. "Prototype Materials Pack." Unity Asset Store.
<https://assetstore.unity.com/packages/2d/textures-materials/prototype-materials-pack-65136> (Accessed Jul. 17, 2020).
- [21] GrigoriyArx. "Wooden Cart." Unity Asset Store.
<https://assetstore.unity.com/packages/3d/environments/fantasy/wooden-cart-65835>
 (Accessed Jul. 17, 2020).
- [22] C. Esteban González, "Diseño e implementación del entorno de videojuego serio de rehabilitación del proyecto "Blexer"," B.S. thesis, Dept. Teoría de la Señal y

Comunicaciones, The School of Telecommunications Systems and Engineering, Technical Univ. of Madrid, Madrid, 2017.

- [23] F. Díez Muñoz, "Informe de prácticas sobre la implementación de elementos diferentes en el videojuego serio Phibys Adventures v2", The School of Telecommunications Systems and Engineering, Technical Univ. of Madrid, Jun 2020.
- [24] Gianni Caratelli. "Mario's way." Freesound.
<https://freesound.org/people/xsgianni/sounds/388079/> (Accessed Jul. 17, 2020).

ANNEX I: Game flowchart with checkpoints in draft form

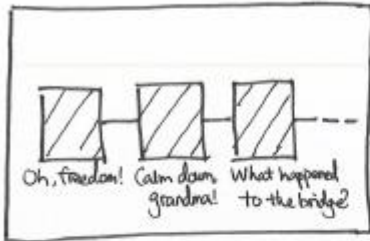




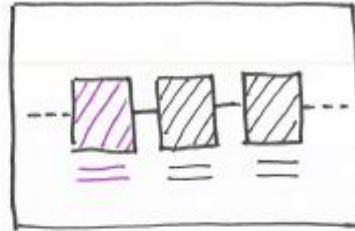
ANNEX II: Advancement Panel draft design

Advancement Panel Interface

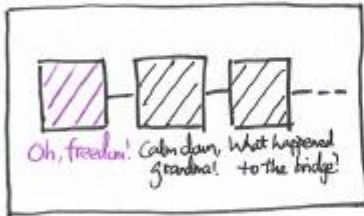
1. Initially



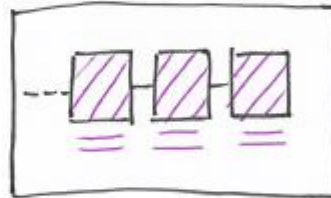
3. Generally



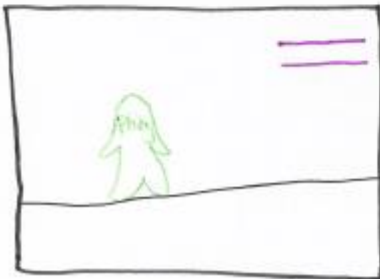
2. After the checkpoint 1



4. When the game is complete



Advancement notifications:



Only text at the right upper right corner.
The background of the texts should be transparent. Players should notice it but not interrupt ~~them~~ be interrupted.

ANNEX III: Budget

In this annex, the budget of the general cost of this project is calculated. Both the used materials cost and the working hours dedicated to this project and this thesis are considered.

- Materials cost:
 - i. Microsoft Kinect 2.0 for Windows: 150 €
 - ii. MSI® P65 Creator laptop (personal): 2000 €
Processor: Intel® Core™ i7-8750H CPU @ 2.20 GHz
Installed RAM: 16.0 GB
System type: 64-bit operating system, x64-based processor
Display adapter: NVIDIA GeForce GTX 1070 with Max-Q Design
Hard disk: 512 GB SSD
- Working hours:
 - i. Pre-learning and preparation: 3 months with 10 hours/week ~ 120 hours in total
 - ii. Developing the project: 5 months with 20 hours/week ~ 400 hours in total
 - iii. Writing the bachelor's degree thesis: 1 month with 50 hours/week ~ 200 hours in total

$$150 + 2000 = 2150 \text{ €}$$

$$120 + 400 + 200 = 720 \text{ hours}$$

Considering that the average income of a telecommunication engineer is about 15 €/hour in Spain, these working hours are equal to 10800 €. Therefore, the final budget should be 12950 €.