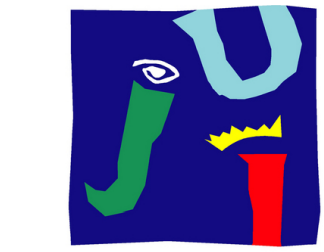

UNIVERSITAT JAUME·I

Games with a Purpose: Developing and Publishing a Video Game about Green Energy

Gonzalo Ramia García

Final Degree Work
Bachelor's Degree in
Video Game Design and Development
Universitat Jaume I

June 19, 2024

Supervised by: José Vte. Martí Avilés, PhD.

To Érica, my always-supporting partner, and Nito and Rengar, my always-meowing cats.

Acknowledgments

First of all, I would like to thank my Final Degree Work supervisor, José Vte. Martí Avilés, PhD, for his guidance.

I also would like to thank Sergio Barrachina Mir and José Vte. Martí Avilés for their inspiring LaTeX template for writing the Final Degree Work report, which I have used as a starting point in writing this report.

Finally, I would like to thank all the friends who helped me during the development of the game, play-testing, giving valuable feedback, and helping me to make decisions. I would like to thank Margarita Gaya in particular, who has always been a supportive friend during stressful times in my university years.

Abstract

The following document is the Final Degree Work for a Bachelor's Degree in Video Game Design and Development. This project aims to rebuild from scratch, expand, and improve the video game A Better Tomorrow to have a complete product ready to be published in an official store.

A Better Tomorrow is a 2D city builder/strategy video game about green energy for touchscreen devices and PCs made with the Unity3D Engine. I originally developed this project in July of 2023 for the Level Up Game Jam. The theme of this competition was Nature and Sustainability. My submission won the Best Use of the Theme award and placed 4th in the public vote.

I firmly believe this project's theme aligns with the values of the Universitat Jaume I. My goal for my Final Degree Work is to overhaul the game entirely and add and improve various game systems, further detailed below.

Keywords

Unity3D, Touchscreen Devices, 2D Vector Art, Green Energy.

Contents

Chapter 1

Introduction

Contents

1.1 Work Motivation

The primary motivation of this thesis is to create a visually appealing, feature-complete game with clean and scalable code about a topic I deeply care about (nature, environment, and green energy). We are currently experiencing a worldwide energy crisis, and I want to spread relevant information about green energy through the game mechanics and theme. I also want to take the opportunity to learn the publishing process of a game in an official store like Google Play.

1.2 Objectives

This project's main goal is to fulfill the abovementioned motivation: creating a project ready to be published. The expected final product is a feature-complete game with beautiful graphics and high attention to detail.

A Better Tomorrow is a game that aims to spread relevant information about green energy with simple yet engaging gameplay. Through its gameplay, the game aims to inform the player about:

- The various types of clean energy recognized by the EU Commission (eolic, solar, hydraulic, gas, nuclear).
- The different structures and generators used to harvest each type of energy.
- The pros and cons of each type of energy.
- The problem of the ever-increasing energy consumption in a finite world.

1.3 Environment and Initial State

A Better Tomorrow is a project developed in a one-week game jam hosted by the Level Up Academy. I co-participated in this game jam with another student of this degree, Érica Masmano Fons. Érica's contribution to the project was making the graphics. I have discussed with her the possibility of using this project as a starting point for my thesis, which she kindly accepted. Her work will not be used.

The current published version of A Better Tomorrow is just a demo/proof of concept of the game systems and themes and should not be considered a representation of the expected results.

As stated before, A Better Tomorrow was developed in a single week. Functionality was prioritized over good, clean, scalable, and reusable code to deliver on time. For this reason, the code quality is way below what is considered acceptable and does not reflect the coding knowledge instructed in this institution. In summary, the current project's technical debt is too high to keep working on.

The plan is to make a new Unity project and rebuild A Better Tomorrow from the ground, completely overhauling all the game's systems. Only the concept/gameplay and some will be saved. Everything else will be remade, improved, and expanded. More precisely, while rebuilding the project, the objectives are:

- Refactoring the code base following Software Engineering principles (SOLID) to have good, clean, scalable code.
- Making the game compatible with touchscreen devices (the game jam version is PC only).
- Implementing a custom, scalable, and reusable localization system that allows the user to switch languages (English, Spanish, and Catalan).
- Adding new mechanics and energy generators.
- Adding a scoring system.
- Improving the UI/UX.

- Improving the player experience with a better game feel and player feedback.
- Redoing and improving some graphics.

1.4 Related subjects

All the knowledge gathered in this degree plays a crucial role in developing this thesis. Nevertheless, the following subjects stand out:

- **VJ1208 Programming II**: This subject teaches students programming concepts using the C# language, Unity's language for scripting.
- **VJ1209 2D Design**: In 2D Design, students learn to use Adobe Illustrator, the tool for making all the graphics of this game.
- **VJ1224 Software Engineering**: This subject's program imparted effective work methodologies for software project organization and development.
- **VJ1227 Game Engines**: This subject teaches the ins and outs of game engines, emphasizing Unity3D.

CHAPTER 2

Planning and resources evaluation

Contents

2.1 Planning

As stated in the previous sections, the main goal of this thesis is to create a visually appealing, feature-complete game with clean and scalable code about a crucial topic (nature, environment, and green energy) while also taking the opportunity to learn the publishing process of a game in an official store like Google Play. With this information in mind, the following plan was created. The plan is divided into four sections: Subject's documentation, Programming, Graphics, and Publishing.

2.1.1 Subject's documentation - Total expected time: 82h

These tasks involve making all the reports or other documents needed for the subject or the correct development of the project.

- **Writing a GDD (10h)**: see Chapter 4 *Game Design Document*.
- **Writing this report (50h)**: This task involves documenting the project's development.
- **Presentation (10h)**: Students have to prepare a presentation for the tribunal.

- **Biweekly reports (12h, two per report)**: for proper student-professor communication, it is required to biweekly report the progress made to the professor supervising the project.

2.1.2 Programming - Total expected time: 200h

- **Rebuilding the game and adding new content (90)**: The most time-consuming development task is rebuilding the game from scratch and re-scripting all the core functionality. This includes:
  - Setting up a repository and the project structure.
  - Creating a placement system.
  - Creating all the different trees, generators, towns, and their logic.
  - Making the gameplay loop.
  - Adding a Weather system and a day-night cycle.
  - Adding a scoring system.
- **Grid System (25h)**: The goal is to create a robust and flexible custom grid system that allows the developer to fully control cell layout, size, and the gap between cells. Additionally, the grid should support basic procedural generation. See section 5.1.1 *Grid System*.
- **Localization System (30h)**: The objective is to create a future-proof and efficient system that allows the developer to add languages simply and conveniently while allowing the user to change languages on runtime. See section 5.2 *Custom Localization System*.
- **Tooltip System (10h)**: This system involves the creation of tooltips that can be dynamically updated and resized on runtime. See section 5.3 *Tooltip System*.
- **Touchscreen and PC support with Unity's New Input System (30h)**: This task involves migrating the game from Unity's Old Input Manager to the New Input System. see section 5.4 *New Input System*.
- **Trophy System (15h)**: Although it was not originally planned, the Trophy System was a last-minute addition that involves the creation of trophies and the logic to store the trophies achieved by the player. See section 5.5 *Trophy System*.

2.1.3 Graphics (Game Art) - Total expected time: 70h

- **Game logo and splash screen (20h)**: This task involves creating promotional art for the game that can be used on menus, banners, social media posts, and other graphics.

- **Interfaces, HUD, and menus (20h)**: The goal is to create a good-looking interface that prioritizes cleanness and ease of use.

- **Environment: tiles, trees, and generators (10h)**: This task encapsulates the creation of most in-game sprites.

- **Game feel (20h)**: This task adds particles, animations, and sound effects that increase player immersion and enhance the gameplay.

2.1.4 Publishing - Total expected time: 5h

- **Inscribing to the Google Play Console Program (1h)**: Needed process to publish in the Google Play Store.

- **Publishing the game in Google Play (4h)**: This task involves submitting the game and setting up the page in their store.

The original plan can be seen in Figure 2.1 Gantt chart.

TASK NAME	Start	Finish	Status	Expected Time	Spent time	Februrary	March	April	May
Subject's documentation				70	86				
Technical Proposal	20/01/2024	26/01/2024	Delivered & Accepted	-	4				
Game Design Document	29/01/2024	18/02/24	Delivered & Accepted	10	12				
Writing the memory	-	-	In Progress	50	70				
Presentation	-	-	Not Started	10					
Biweekly reports	-	-	In Progress	1h per report	3				
Rebuilding the game (Programming)				91	243				
Setting up a GitHub repository and Project Structure	-	-	Done	1	1				
Placement system	-	-	Done	15	10				
Structures: Trees, Generators and Towns	-	-	Done	25	10				
Gameplay Loop	-	-	Done	25	10				
Weather system, Daynight Cycle	-	-	Done	10	5				
Designing the scoring system	-	-	Done	3	3				
Implementing the scoring system	-	-	Done	10	12				
Data persistence: save player's score	-	-	Done	2	1				
Grid System			DONE	15	25				
Localization (Programming)			DONE	30	8				
Research about how localization systems work	-	-	Done	5	5				
Implementing a custom localization system	-	-	Done*	25	3				
Tooltip System			DONE	10					
Touchscreen support (Programming)			DONE	40	75				
Research about Unity's new input system	-	-	Done	10	15				
Implementing input for both PC and Android	-	-	Done	30	60				
Graphics (Art)				30	78				
In-game HUD	-	-	Done	10	20				
Improving the sky/backgrounds	-	-	Done	4	4				
Menus	-	-	Done	10	20				
Vegetation	-	-	Done	2	12				
Structures	-	-	Done	2	20				
Tiles	-	-	Done	2	2				
Game feel and polish (Art)				20	25				
Particles and VFX	-	-	Done	10	10				
Adding sounds	-	-	Done	5	15				
Adding animations	-	-	Not Started	5	0				
Publishing				5	0				
Inscribing myself to the Google Play Console Program	-	-	Not Started	1					
Publishing the game	-	-	Not Started	4					
Total				311	507				

Figure 2.1: Original planning Gantt Chart.

2.2 Deviations from the original plan

The development start was rough because the Desktop PC power supply broke in early 2024. There were several issues with delivery, and the PC was unavailable until March.

During the development of this project, coding skills and the time required to write clean code were underestimated. Conversely, the ability to create visually appealing graphics was extremely overestimated.

One of the goals of this project was to create a beautiful game. Game art is probably the most crucial pillar to developing a successful game, a key aspect of a marketable product. Humans are inherently visually driven; visual appeal determines whether a potential player engages with a video game. A gorgeous game captures attention.

This is why games tagged as *cozy* are rising [5]. Cozy games prioritize aesthetics over gameplay and are designed to evoke feelings of comfort with charming visuals and stress-free environments.

A Better Tomorrow aims to be a cozy game, but producing high-quality graphics is challenging and requires significant time. During the development of this project, game art emerged as a particular Achilles heel. The estimated time for creating graphics was considerably inaccurate, while most programming tasks were completed somewhat ahead of schedule.

As anecdotal evidence, creating the graphics for the trees took almost twice as long as developing the Localization System.

Overall, the total time spent on game development was slightly higher than anticipated but still within a reasonable range. Most of the art-related tasks took twice as long as initially planned.

2.3 Resource Evaluation

To properly execute any planning, evaluating the available resources and estimating the costs is crucial. This includes both human resources and the tools needed to do the job.

The estimated cost of this project is **11427,19€**. This estimation does not consider all the hours and efforts put into creating the first iteration of A Better Tomorrow when the developers participated in the Level Up Game Jam.

2.3.1 Human Resources

The average salary of a junior developer oscillates between 18000 and 26000€, equivalent to around 1800€ gross per month[3]. The development cycle of this project is four months, which adds up to 7200€.

2.3.2 Hardware

The following workstation was used to develop the video game:

- **Desktop PC**:
  - Motherboard: Asus PRIME B560M-A (112.99€).
  - CPU: Intel i5-10400f LGA1200, 2.9 GHz, 12MB cache (104.90€).
  - GPU: GeForce RTX 3060 Gaming OC 12G (309.90 €).
  - RAM: 2x8GB DDR4-300 G.SKILL Ripjaws (56.99 €).
  - PC-case and power supply: Nox Infinity Omega ARGB USB 3.0 (39.99€).
- **Monitor**: BenQ ZOWIE XL2411P (189.99€).
- **Mouse**: GXT 109 FELOX (14.99€).
- **Keyboard**: Ozone CHERRY MX TKL STRIKE BATTLE Mechanical Keyboard (89.90€).
- **Microphone**: HyperX QuadCast S (179.99€).

The workstation for this project costs around 1127.52€

2.3.3 Software

For the development of this project, the following software was used:

- **Unity3D**[14]: Unity is a well-established general-purpose engine that fits with the goals of this project. Using Unity's Universal Render Pipeline, developers can create good-looking games that work fine on PCs and low-end touchscreen devices.
- **JetBrains Rider IDE**[10]: JetBrains IDEs are one of the most complete IDEs available, and using Rider's tools will drastically increase productivity and code quality. A standard Rider license costs 149,00€ the first year, but student licenses are free.
- **Odin Inspector**[11]: This asset is a Unity plugin that allows programmers to tidy up the inspectors using attributes easily. It makes testing and designing easier. It cost 54,80€ per seat (one-time payment).

- **All in 1 Shader**[13]: A tool to add shader effects to sprites in Unity quickly. It costs 34,90€ per seat (one-time payment).
- **Adobe Illustrator**[1]: The industry standard application to create vector graphics. This software costs 316,97€ per year.
- **GitHub Desktop**[6]: Free software to comfortably work with code hosted on GitHub.
- **Trello**[2]: Free online tool for planning and organizing.
- **Grammarly**[8]: Enhanced spellchecker to improve English writing. The plan used costs 144€/year (12€/month).
- **Overleaf**[12]: An online free collaborative tool for writing, editing, and publishing documents in LaTeX.

The licenses needed for the development of this project add up to 699,67€

2.3.4 Infrastructure

Other costs that should be considered when developing a project are renting an office and electricity.

According to the research, an office can spend between 52,5 kWh and 110,6 kWh per square meter per month[19]. In Castellón de la Plana, renting an office costs an average of 500 euros per month. [7]. The office should be rented for at least four months (February, May, April, and March).

With all these considerations, the infrastructure needed to develop this project could add an extra 2400€. Bear in mind that this is no more than an educated guess. Energy prices fluctuate, and the size of the office matters. Also, the number of workers in an office plays a crucial role in energy consumption.

CHAPTER 3

System Analysis and Design

Contents

This chapter presents the proposed work's requirements analysis, architecture, and interface design.

3.1 Requirement Analysis

3.1.1 Functional Requirements

A functional requirement defines a function of the system that will be developed. This function is a set of inputs, behaviors, and outputs. The functional requirements of this application are:

Main Menu Screen Requirements

- **R1.** When the game has finished loading, the player can start the game by clicking/tapping anywhere on the screen. (see Table 3.1).
- **R2.** The player can quit the game by pressing the *Escape key/Android Back Arrow*. (see Table 3.2).

- **R3.** The player can start a new game by pressing the *Play button*. (see Table 3.3).
- **R4.** The player can access the challenges window by pressing the *Challenges button*. (see Table 3.4).
- **R5.** The player can access a set of slides that provide information about the various types of energy by clicking/tapping the *Learn button*. (see Table 3.5).
- **R6.** The player can open the settings window by clicking/tapping the *Options button* (cog icon). (see Table 3.6).
- **R7.** The player can access an about window by clicking/tapping the *Information button* (i icon). (see Table 3.7).
- **R8.** The player can access their trophy collection by clicking/tapping the *Trophies button* (cup icon). (see Table 3.8).

Game Screen Requirements

- **R9.** The player can open the settings *Pause Menu* by clicking/tapping the *Pause button/Escape key/Android Back Arrow*. (see Table 3.9).
- **R10.** The player can go back to the *Main Menu* by clicking/tapping the *Return to menu button*. (see Table 3.10).
- **R11.** The player can select a structure by clicking/tapping the structure buttons on the hotbar at the bottom of the screen. (see Table 3.11).
- **R12.** The player can place a structure on a non-occupied tile by clicking/tapping on it with any structure selected. (see Table 3.12).
- **R13.** The player can enter and exit *Remove Mode* by clicking/tapping on *Remove Mode Button* on the hotbar found at the bottom of the screen. (see Table 3.13).
- **R14.** The player can remove a structure by clicking/tapping on an occupied tile in *Remove Mode*. (see Table 3.14).
- **R15.** The player can provide energy to towns by clicking/tapping on it when they demand energy. (see Table 3.15).
- **R16.** The player will lose the game if it doesn't provide energy to the towns or if the environmental health goes below zero. (see Table 3.16).

3.1.2 Non-functional Requirements

Non-functional requirements encompass criteria impacting a system's functioning, deployment, and upkeep, spanning performance, reliability, scalability, security, usability, and maintainability. It's crucial to keep these in mind throughout the system's development to ensure it hits the mark in terms of quality.

- **R17.** The system needs 90 MB of storage to install the game.
- **R18.** The system should store non-sensitive player data, such as the selected language, their current trophies, or other settings.
- **R19.** The system will be user-friendly, prioritizing ease of use and readability with clear interfaces.
- **R20.** Any errors during the execution of the game should be notified to the player.

3.2 System Design

This section showcases the system's logical and operational design. It outlines various use cases derived from the functional requirements. Subsequent pages detail these use cases, accompanied by the Use Case diagram (see Figure 3.1) and Activity diagram (see Figure 3.2).

Requirement:	R1
Actor:	Player
Description:	The player can start the game by clicking/tapping anywhere on the screen
Preconditions:	The game has finished loading
Normal sequence:	1. The player clicks/taps the screen 2. The player accesses the *Main Menu*.
Alternative sequence:	None

Table 3.1: Case of use "R1"

Requirement:	R2
Actor:	Player
Description:	The player can quit the game by pressing the *Escape key/Android Back Arrow*.
Preconditions:	The player has to be on the *Main Menu*
Normal sequence:	1. The player attempts to quit the game 2. A window shows up, asking for confirmation 3. The player confirms quitting the application
Alternative sequence:	The player cancels quitting, hiding the window and returning to the *Main Menu*

Table 3.2: Case of use “R2”

Requirement:	R3
Actor:	Player
Description:	The player can start a new game by pressing the *Play button*
Preconditions:	The player has to be on the *Main Menu*
Normal sequence:	1. The player presses the button 2. The loading screen shows up 3. The loading finishes and the scene changes to the *Game Screen*
Alternative sequence:	None

Table 3.3: Case of use “R3”

Requirement:	R4
Actor:	Player
Description:	The player can access the challenges window by pressing the *Challenges button*
Preconditions:	The player has to be on the *Main Menu*
Normal sequence:	1. The player presses the button 2. The challenges window shows up 3. The player selects a challenge 4. The loading screen shows up 5. The loading finishes and the scene changes to the *Game Screen*
Alternative sequence:	The player closes the challenges window

Table 3.4: Case of use "R4"

Requirement:	R5
Actor:	Player
Description:	The player can access a set of slides that provide information about the various types of energy by clicking/tapping the *Learn button*
Preconditions:	The player has to be on the *Main Menu*
Normal sequence:	1. The player presses the button 2. The learn window shows up 3. The player can move through the different tabs of this window
Alternative sequence:	The player closes the Learn window

Table 3.5: Case of use "R5"

Requirement:	R6
Actor:	Player
Description:	The player can open the settings window by clicking/tapping the *Options button* (cog icon)
Preconditions:	The player has to be on the *Main Menu*
Normal sequence:	1. The player presses the button 2. The settings window shows up 3. The player can modify settings such as the language
Alternative sequence:	The player closes the Settings window

Table 3.6: Case of use "R6"

Requirement:	R7
Actor:	Player
Description:	The player can open the settings window by clicking/tapping the *information button* (i icon)
Preconditions:	The player has to be on the *Main Menu*
Normal sequence:	1. The player presses the button 2. The About window shows up
Alternative sequence:	The player closes the About window

Table 3.7: Case of use "R7"

Requirement:	R8
Actor:	Player
Description:	The player can open the settings window by clicking/tapping the *Options button* (cog icon)
Preconditions:	The player has to be on the *Main Menu*
Normal sequence:	1. The player presses the button 2. The settings window shows up 3. The player can modify settings such as the language
Alternative sequence:	The player closes the settings window

Table 3.8: Case of use "R8"

Requirement:	R9
Actor:	Player
Description:	The player can open the *Pause menu* by clicking/tapping the *Pause button/Escape key/Android Back Arrow.*
Preconditions:	The player has to be on the *Game Screen*
Normal sequence:	1. The player presses the button/key 2. The game pauses, and the *Pause Menu* shows up
Alternative sequence:	The player closes the pause menu, unpausing the game

Table 3.9: Case of use "R9"

Requirement:	R10
Actor:	Player
Description:	The player can return to the Main Menu by clicking/tapping the *Return to menu button.*
Preconditions:	The player has to be on the *Pause Menu*, inside the *Game Screen*
Normal sequence:	1. The player presses the button/key 2. The loading screen shows up 3. The loading finishes, and the scene changes to the *Main Menu*
Alternative sequence:	The player unpauses the game, exiting the pause menu

Table 3.10: Case of use "R10"

Requirement:	R11
Actor:	Player
Description:	The player can select a structure by clicking/tapping the structure buttons on the hotbar at the bottom of the screen
Preconditions:	The player has to be on the *Game Screen*
Normal sequence:	1. The clicks/taps on any structure icon 2. The structure gets selected
Alternative sequence:	None

Table 3.11: Case of use "R11"

Requirement:	R12
Actor:	Player
Description:	The player can place a structure on a non-occupied tile by clicking/tapping on it
Preconditions:	1. The player has to be on the *Game Screen* 2. The player has to have a selected structure 3. The tile(s) has to be free 4. The player has enough energy 5. The environment is healthy enough
Normal sequence:	The structure gets placed
Alternative sequence:	If the structure cannot be placed because of preconditions 3, 4, or 5, a warning message will appear explaining the reason why the structure could not be placed

Table 3.12: Case of use “R12”

Requirement:	R13
Actor:	Player
Description:	The player can enter and exit *Remove mode* by clicking/tapping on *Remove Mode button* on the hotbar found at the bottom of the screen
Preconditions:	The player has to be on the *Game Screen*
Normal sequence:	1. The clicks/taps on the *Remove Mode button* 2. The player enters *Remove Mode*
Alternative sequence:	If the player is already on *Remove Mode*, pressing this button exits the *Remove Mode*

Table 3.13: Case of use “R13”

Requirement:	R14
Actor:	Player
Description:	The player can remove a structure by clicking/tapping on an occupied tile
Preconditions:	1. The player has to be on the *Game Screen* 2. The player has to be on *Remove Mode* 3. The tile(s) has to be occupied
Normal sequence:	The structure gets removed
Alternative sequence:	If preconditions are not met, nothing happens

Table 3.14: Case of use “R14”

Requirement:	R15
Actor:	Player
Description:	The player can provide energy to towns by clicking/tapping on it when they demand energy
Preconditions:	1. The player has to be on the *Game Screen* 2. The player has clicked on a demanding energy town 3. The player has enough energy
Normal sequence:	The town receives the energy and stops demanding energy
Alternative sequence:	If precondition three is not met, a warning "You have no energy" shows

Table 3.15: Case of use “R15”

Requirement:	R16
Actor:	Player
Description:	The player will lose the game if it does not provide energy to the towns or if the environmental health goes below zero
Preconditions:	1. The player has to be on the *Game Screen*
Normal sequence:	1. The player does not provide energy or does not take care of the environment 2. The game ends 3. The summary screen shows up
Alternative sequence:	None

Table 3.16: Case of use “R16”

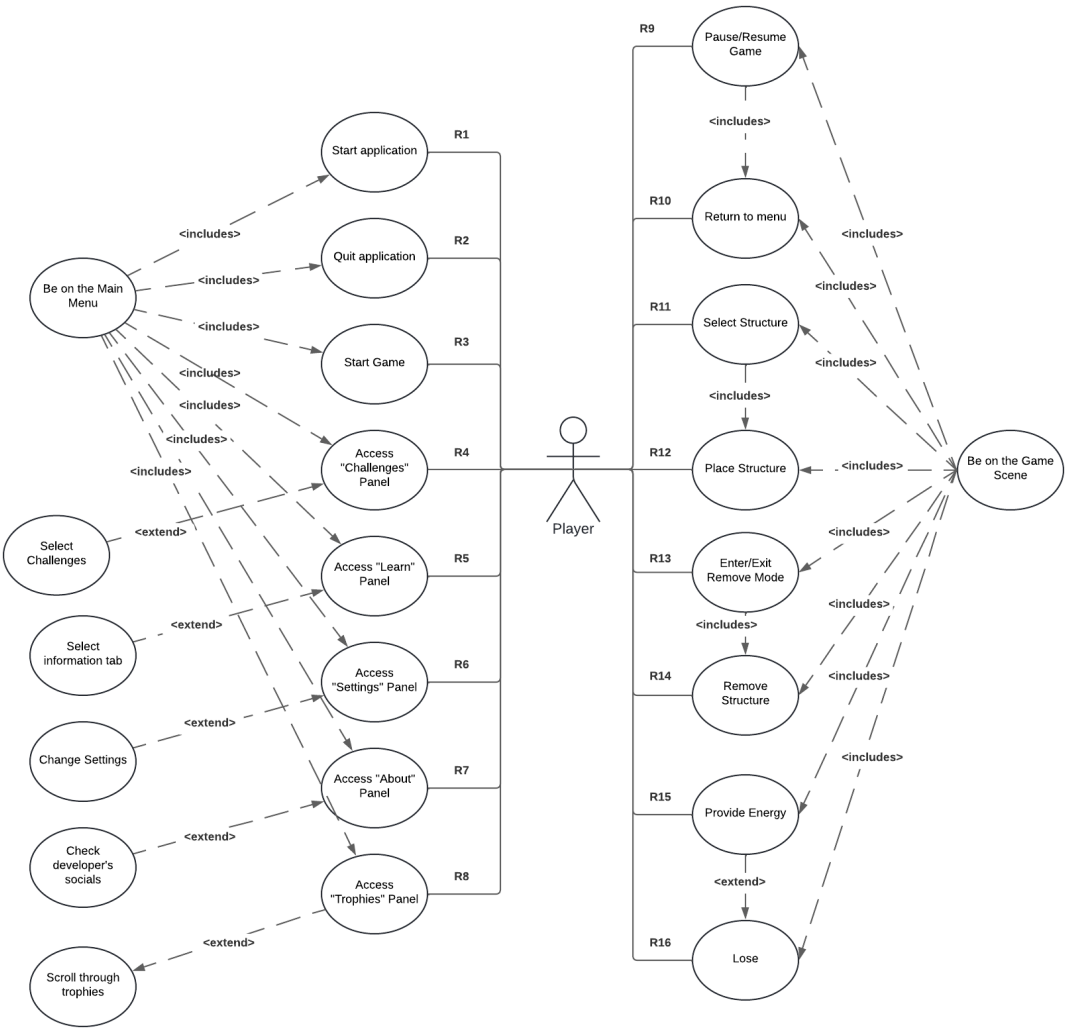

Figure 3.1: Case use diagram.

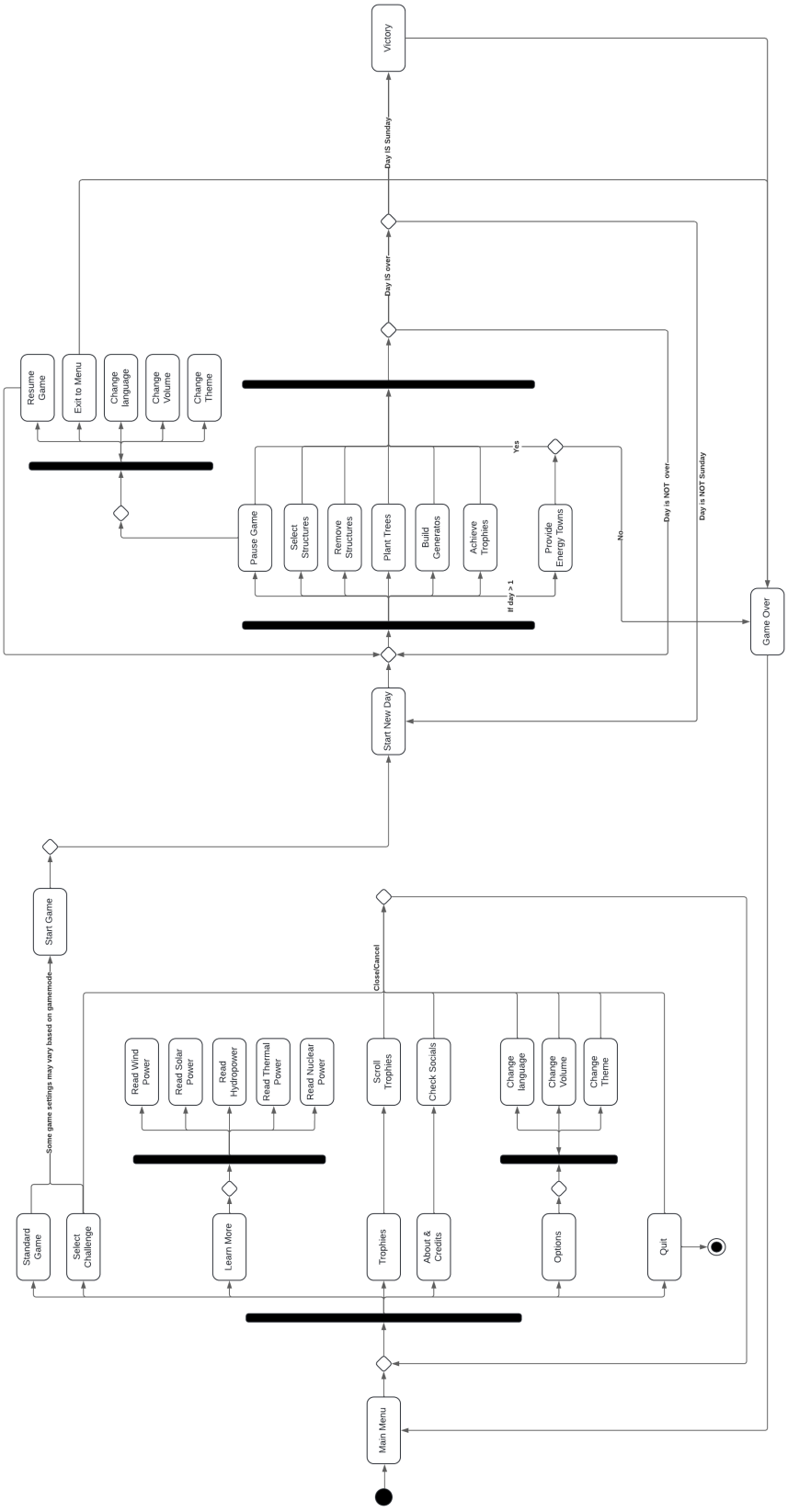

Figure 3.2: Activity diagram.

3.3 System Architecture

The application has been tested on various devices, from phones to desktop PCs. The game is guaranteed to run at least 30 FPS on any Android device that meets the following requirements or higher.

- **RAM**: 6GB.
- **CPU**: Helio G35 Octa Core, 2.3 GHz.
- **OS**: Android 12.

Since the app has been optimized for low-end Android devices, most modern desktop PCs should be able to run the game at high FPS without any issues. The PC used for benchmarking has the following hardware:

- **CPU**: Intel i5-10400f LGA1200, 2.9 GHz, 12MB cache.
- **GPU**: GeForce RTX 3060 Gaming OC 12G.
- **RAM**: 2x8GB DDR4-300 G.SKILL Ripjaws.

This machine was able to reach 200+ FPS without issues.

3.4 Interface Design

The game's menus and HUD will be minimalist, clean, and simple. They will feature simple, monochromatic icons and diagrams. The fonts will be sans serifs and rounded. The goal is to create menus that are easy to read, understand, and navigate. Some of the following images are mock-ups or works in progress (WIP) and should not be considered definitive.

On the Main Menu UI, a panel with some text buttons will be found on the left. Some icon buttons will be found on the top right. Clicking on the icon buttons will unfold a panel on the left. The trophy icon has been clicked in Figure 3.1, revealing the trophy expositor. A confirmation window will appear if the user attempts to quit (see Figure 3.2).

On the Game Screen (see Figure 3.3), the player can find key information like the current day, hour, and weather on the top of the screen. On the left, environmental health can be found. On the bottom, the hotbar with all the structures. Additionally, a button in the top left corner pauses the game, revealing the pause menu (see Figure 3.4).

While the game loads, a nice scrolling seamless background will be shown. A loading progress bar will communicate to the player the loading progress. As it loads, this

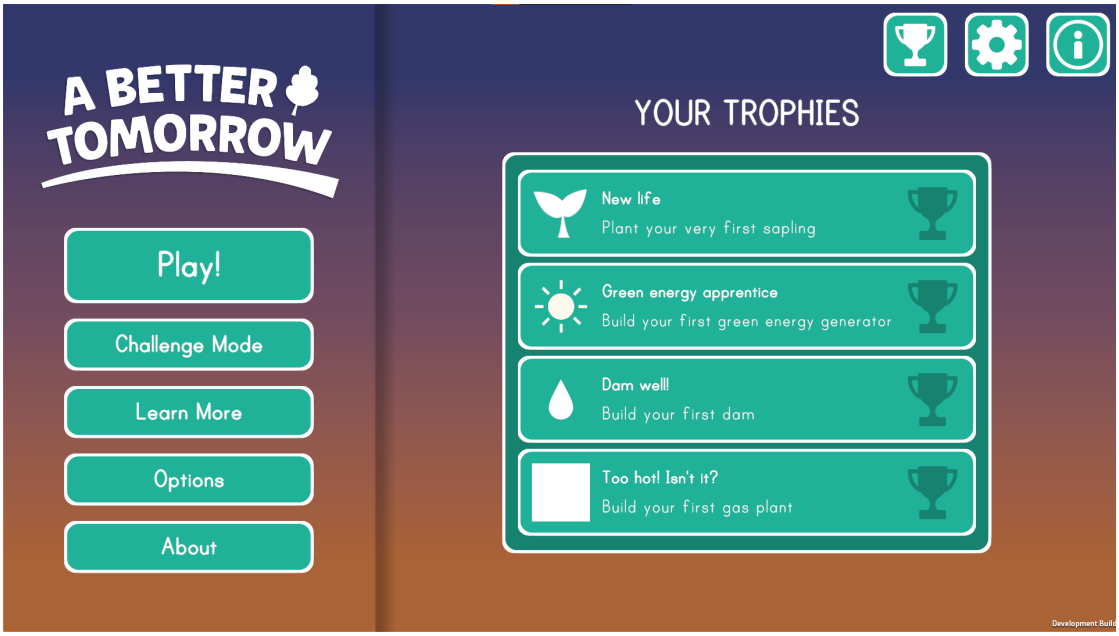

Figure 3.3: Main menu UI.

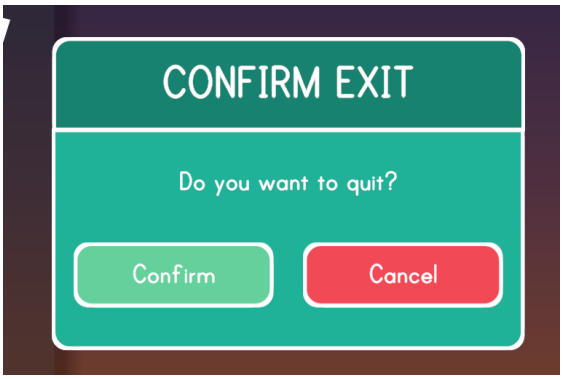

Figure 3.4: Exit window.

progress bar shows trees, windmills, houses, and other in-game sprites (see Figure 3.5).

Some UI buttons will show a dynamic tooltip (see section 5.3 *Tooltip System*). This tooltip gives the player contextual information in a non-intrusive way. Tooltips have content and an optional header(see Figure 3.6).

Figure 3.5: Game Screen.

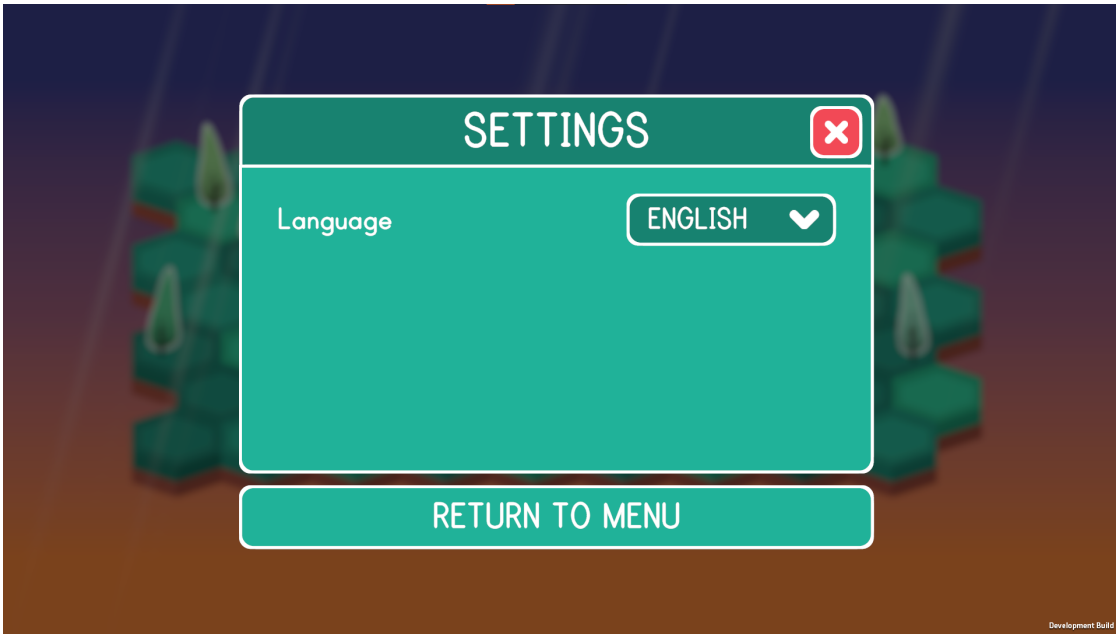

Figure 3.6: Pause UI.

Figure 3.7: Loading screen.

Figure 3.8: Tooltip example.

CHAPTER 4

Game Design Document

Contents

In this chapter, the content of the game's Game Design Document (GDD) can be found. Writing a GDD is essential for the adequate development of a video game. A well-written GDD helps with organization, planning, scope, and consistency.

4.1 General Data

Project name: A Better Tomorrow.
Platforms: Android, PC.
Target Audience: Everyone, but mainly kids and young teenagers (7 -16 years old).
PEGI: 3

4.1.1 Summary

Energy is essential. Everything in this world depends on energy. In the context of societal and economic development, energy propels our industries, technological advancements, and civilization's progress.

Energy production/consumption and the environment are two topics tightly tied together. We need to produce energy for our industrial activities, but in the process, we have been harming the environment (emissions) and exploiting its resources (non-renewable energy sources, like coal and oil) for centuries.

This approach is not sustainable, and humans have been trying to find a solution to the sustainability problem for decades. That's why, in recent years, most countries have been transitioning to renewable energy, also known as green energy. Renewable energy is energy derived from natural resources that are virtually inexhaustible. Obtaining energy from these sources also creates far lower emissions.

We are currently experiencing a devastating environmental and energy crisis. The energy price in the EU reached record levels in both 2022 and 2023. Since the EU imports much of its energy, prices are quite volatile, but they usually do not vary higher than 30% during a year. In 2022, prices more than doubled (source: European Council link it!). There are many reasons for this, namely the Russian invasion of Ukraine or the increasing temperatures and constant heat waves.

The EU has been trying to destigmatize gas and nuclear power for years to combat the energy crisis. On July 6, 2022, the EU Parliament labeled gas and nuclear power as green, equating it with renewables. This controversial decision caused heavy backlash from several political groups.

A Better Tomorrow is a game that aims to spread relevant information about green energy with simple yet engaging gameplay. Through its gameplay, the game aims to inform the player about:

- The various types of clean energy recognized by the EU Commission (eolic, solar, hydraulic, gas, nuclear).
- The different structures and generators used to harvest each type of energy.
- The pros and cons of each type of energy.
- The problem of the ever-increasing energy consumption in a finite world.

4.2 Gameplay

4.2.1 Gameplay summary

A Better Tomorrow is a 2D city builder-style environmental video game about clean energy. The player's goal is to last for an in-game-time week, providing clean energy to the citizens while respecting the environment. To achieve this goal, the player has to manage three resources: energy, the environment's health, and space.

Energy is the primary resource of the game. It has two uses: satisfying the energy demands of cities and villages and building new clean energy generators. These generators provide energy over time. The production rate of the generator varies depending on the current weather (e.g., a solar panel is more efficient under sunny weather conditions, while windmills excel in windy and stormy weather).

There are five energy generators: windmills, solar panels, dams, gas plants, and nuclear plants (on July 6, 2022, the UE Parliament labeled gas and nuclear power as green, equating it with renewables). Each generator has an energy cost, an energy production rate (that may vary depending on the weather), and an environmental cost (i.e., how harmful it is for the environment to build the structure). These structures appear in the toolbar at the bottom of the screen. The player has to drag a structure from the toolbar and drop it onto the gameboard to build it, paying its cost.

The environment has a health bar. Whenever the player builds a new generator, the environment takes damage (the environmental cost value of the generator). The player cannot keep building if the environment's health is too low. The environment can heal by plating saplings that function as any other structure. The saplings eventually flourish into trees, which grants extra environmental health.

The game occurs on a procedurally generated board. The board always has the same amount of tiles. A random number of tiles are generated with trees on them. New cities and villages spawn on the board when an in-game day goes by.

Cities and villages occupy one tile and demand energy every few seconds. The player must provide the required energy by tapping/clicking on the city/village in a given period. The game is over if the timer runs out and the player has not provided the energy to the citizens.

Therefore, each in-game day, the player has to provide more energy while having less space to plant trees and build generators, exponentially increasing the game's difficulty. The game ends when the player manages to satisfy the citizen's energy demand for a whole in-game week.

After beating the game, the player unlocks the Challenge Mode. A Better Tomorrow base game prioritizes sending a message over gameplay. On the other hand, Challenge Mode is designed for hardcore players who enjoy the strategy and management part of the game. Challenge Mode is an endless game mode with two additional extreme weather conditions caused by global warming: freezing temperatures and heat waves. These extreme weather conditions can shut down generators (e.g., a dam won't' generate electricity in a heat wave).

4.2.2 Rules and mechanics

The board comprises seven rows x nine columns of hexagonal tiles. When a new game starts, the board is procedurally filled with trees. Trees and tiles' color palettes will be chosen randomly when the game starts to add visual variety.

Each tile can have up to one *structure* (in the context of this game, a Structure is a game entity that fills a tile) on it. If a tile has any entity on it (like a tree), it is considered occupied, and the player is not allowed to build in said tile. Players can free tiles using the axe on their toolbar. If freeing a tile implies chopping down a tree, the environment will suffer ten hit points of damage.

The player starts the game with 150 energy. If the player has enough energy, they can drag structures from their toolbar to build them on any free tile. Each structure has an energy cost, an environmental impact, a production rate, a production amount, a list of favorable weather conditions, a list of detrimental weather conditions, a favorable weather multiplier, and a detrimental weather multiplier.

The player starts the game on Monday. Initially, only saplings, windmills, and solar panels are unlocked. On Wednesday, the player gets access to dams. On Thursday, the gas plants become available. Finally, the nuclear plant is unlocked on Friday.

Each day, the weather is randomly selected. The same weather can be selected twice in a row. It is planned to implement the following weather: Sunny, Windy, Rainy, Cloudy, and Heatwave.

When a day elapses, new cities and villages spawn. If they cannot find any free tile to spawn, they randomly select one, which may destroy a player-build generator or trees. If trees are destroyed, the environment will be hurt.

Villages and cities are essentially negative energy generators. When their bar fills, they demand energy from the players instead of producing it. The game ends if the player does not provide the required amount in 10 seconds. Every day after Wednesday, a random existing village will evolve into a city. This table shows the number of villages and cities active each day.

When an in-game week passes, the player is rewarded with a victory screen, and the game ends, unlocking the Challenge Mode. Challenge Mode works exactly like standard mode, but after day 7, instead of spawning new cities, the current cities' energy demand increases.

Therefore, each in-game day can be interpreted as a level, and the player's goal is to overcome seven levels (Monday, Tuesday, Wednesday, Thursday, Friday, Saturday, and

Day	Active Villages	Active Cities
Monday	0	0
Tuesday	2	0
Wednesday	3	1
Thursday	4	2
Friday	3	4
Saturday	2	6
Sunday	1	8

Sunday) to win. Additionally, the player will get a score when the game ends based on their performance.

4.3 Controls and camera

The game is meant to be played exclusively with the mouse. If the user hovers over any UI icon or tile entity, the dynamic custom tooltip system that will be implemented will provide some context for the user. To place structures, the player can drag and drop from the toolbar or select a structure by clicking on its icon and then clicking on any free tile on the game board.

For touchscreen devices, the user's finger acts as the mouse, and tapping the screen performs the same actions as left-clicking.

4.3.1 PC Exclusive features

Keyboard shortcuts will be added for PC players. For example, the ESC key will be mapped to act as clicking the pause button in the UI, and number keys will be mapped to select the UI elements of the Hotbar.

4.3.2 Camera

The camera is stationary: The player can't move it, and it always shows the whole game board.

4.4 Graphics and Styling

4.4.1 Style

A Better Tomorrow will follow minimalist vector-style art. The project aims to evoke a sense of calmness and tranquility. Therefore, the in-game graphics will have simple, clean shapes and plain colors. The color palette will feature a variety of soft turquoise greens and indigo blues. The flora colors will vary in each game, adding some variety.

(e.g., autumn colors) The goal is to create a game with a similar general aesthetic to Dorfromantik[9]. Although this game doesn't use vector-style graphics, its feel and art direction are similar.

4.4.2 Interfaces (HUDs and menus)

The game's menus and HUD will follow the same minimalist, clean, and simple style previously described. They will feature simple, monochromatic icons and diagrams. The fonts will be sans serifs and rounded. The goal is to create menus that are easy to read, understand, and navigate. These types of interfaces were popular in the early 2000s.

4.4.3 Miscellaneous

Using Unity's particle various visual effects will be implemented to increase player immersion. For example, to visually show the current weather, "god rays" will appear on the screen during sunny weather, water droplets will splash on the screen when it rains, and some generators will emit particles when they produce energy (e.g., Nuclear plants will throw steam off their chimneys).

4.5 Audio, Music and Sound Effects

The music featured in the game will be a relaxing, calm track. It is planned to add copyright-free music found online with a similar vibe as Animal Crossing OST will be implemented.

The ambient sound effects (rain, wind, bird noises, etc) will also be copyright-free SFX found online. On the other hand, it is planned to record the non-diegetic sounds of this project (like placing a structure or hovering over a tile).

CHAPTER 5

Work Development and Results

Contents

Properly fitting, given the project's name, during the development of this project, there was a vision: future-proof. The objective was not to create the game but a clean code base, creating scripts and systems that are reusable and scalable, which could be dragged into any future project and work as expected. This implies that all systems must be tightly decoupled and go beyond what is just needed for the project.

5.1 Rebuilding the Game

5.1.1 Grid System

A Better Tomorrow has a board of seven rows x nine columns of hexagonal tiles. When a new game starts, the board is procedurally filled with trees. The script to generate this grid could be a couple of for-loops and a random chance to instantiate a tree. However, that was not the goal. The goal was to build a well-made, scalable, and flexible grid system. Unity's grid system is half-baked and lacks many options desired. For example, Unity's grid class does not allow setting a gap between cells if the layout is hexagonal

and only supports pointed-top hexagonal grids, not flat-top.

The tool creation process for Unity has proven to be a captivating journey of learning and development as a programmer. One has come to appreciate the importance of dedicating additional time to constructing a resilient, expandable system rather than merely addressing the immediate requirements of a particular scenario. This change in perspective has notably improved the development approach.

As a result of these efforts, a versatile and highly functional grid generator tool was created. It supports the creation of rectangular, isometric, flat-top hexagonal, and pointed-top hexagonal grids. Users can adjust the scale and spacing of each tile, and it works seamlessly in both 2D and 3D environments. The tool operates through prefab instantiation, allowing for easy customization of behavior-specific tiles inheriting from the tile base class.

The new tool enabled experimentation with map generation and testing various layouts. It was clear that using procedural generation to make the map more attractive could be a great addition. A cellular automaton is the simplest yet most efficient way to implement procedural generation for tile-based games.

A cellular automaton (C.A.) is a mathematical and computational model for a dynamic system that evolves in discrete steps called generations. It has the following elements:

- A regular space composed of cells
- A finite amount of states that each cell can take
- An initial state
- A set of evolution rules

In a tile-based game, the regular space is the grid, and the cells are the tiles. The cell states are the types of tiles supported. As the initial state, it was decided to fill the grid randomly based on the fill percentage variable. This C.A. uses Moore Neighbourhood [18] and a fully asynchronous update [16]. The evolution rules are pretty simple: if a cell is surrounded by more cells with a state of 0 than 1, it changes its state to 0 and vice versa. It remains unchanged if surrounded by the same number of tiles in each state. (See Appendix B: Source Code)

Using an auxiliary class, the C.A. was serialized into Unity's inspector and added to the Grid Generator tool. Various configurations were explored to enhance the map generation in A Better Tomorrow. For example, water and lake tiles were introduced, restricting the construction of specific structures nearby. (For example, Players can only build a dam near a water tile.) Although this concept had potential, the map's small

size, designed for mobile devices, made the game less fun.

Nevertheless, the C.A. will be used to instantiate the trees. With this approach, the tiles with trees will tend to be near other tree tiles, making the generation more organic.

5.1.2 Structures

In the context of this video game, a structure is any game entity that can occupy a tile or group of tiles. All structures inherit from a base class named StructureBase.cs.

Structures can be divided into two major groups: dynamic and static. Each dynamic structure has an energy cost, an environmental impact, a production rate, a production amount, a list of favorable weather conditions, a list of detrimental weather conditions, a favorable weather multiplier, and a detrimental weather multiplier. Dynamic structures have a progress bar and perform an action when their progress bar fills. This group includes generators, saplings, and towns. When sapling's bar fills, they grow into trees. When the generators' bar fills, they produce energy.

Towns are quite similar to generators, although they work slightly differently. Towns are essentially negative energy generators. Their bar does not fill but rather empties. When their bar is empty, they start demanding energy. Villages also have special logic to evolve into cities when a day elapses.

The only static structures are trees. Trees do not perform any action and thus lack a progress bar. The only action trees perform is adding 10 to the environmental health on spawn.

5.1.3 Weather System

A key gameplay feature in A Better Tomorrow is the weather system. This system works with an Enum with the [Flags] attribute. Therefore, each weather represents a bit. (see Table 5.1)

With this approach, bitwise operations can combine flags (bitwise OR) and check for specific flags (bitwise AND). This allows for the easy creation of new structures. For example, a structure that reacts to cloudy and rainy weather would have its weather flag set to 0000 0011. When the weather changes, an event is thrown. Every object subscribed to that event can check if they have the current weather flag with a simple bitwise AND. If the result of the operation != 0, they should react to the current weather.

Weather is selected randomly when a day elapses. The available weather conditions vary depending on the game mode. When a new weather is selected, an adequate VFX and SFX play to increase player immersion.

Weather	Decimal	Binary
Cloudy	1	0000 0001
Rainy	2	0000 0010
Sunny	4	0000 0100
Windy	8	0000 1000
Snowstorm	16	0001 0000
Heatwave	32	0010 0000
Lightning Storm	64	0100 0000

Table 5.1: Weather values

5.2 Custom Localization System

One aspect that differentiates a published and completed video game from a simple prototype is localization. Localization is the process of adapting content to fit a specific audience. Beyond just translating, localization requires sensitivity to cultural nuances, idiomatic expressions, humor, symbols, etc. In a globalized world, localization allows people from different languages and cultures to enjoy the experience and connect. This is especially true for games with an informative intention, like A Better Tomorrow. The objective was to create a future-proof Localization System that would be usable in forthcoming projects.

5.2.1 Why develop a custom solution?

Unity already has a localization tool (although pretty limited), but creating custom tools allows developers to understand better how things work. It gives developers more control over the systems, allowing more fine-tuning. Additionally, it enhances the learning experience that is the development of this Bachelor's Thesis.

5.2.2 Research and Objectives

After researching and finding many valuable resources, the intention was to combine the concepts learned into a single system that fits the needs of this project. Most localization systems found online were either too overcomplicated for this game or lacked the optimizations to quickly change languages on runtime, a feature desired. The goal was to make a system that was:

- **Scalable**: Adding new languages is as convenient as possible
- **Simple**: Simple to implement and modify
- **Future-proof**: Adding this system in future projects is as simple as importing a Unity package.
- **Efficient**: Fast enough to allow users to change the language on runtime.

- **Feature complete**: Supports "smart text"

Most localization systems use a datasheet file (e.g., a .csv file). Each column in the file is a language, and each row is a text value that must be localized. The first column of the file has the text keys. When the game boots, given a language, the system parses this file and generates a dictionary with the key-text value pair given the language. Then, in-game, each text element has a key, and the string is changed based on the value associated with the key.

This approach is adequate but has a problem: The file can quickly become massive. This makes adding new languages inconvenient and error-prone since the developer has to manage a vast file to add new entries/languages. Moreover, rebuilding the key-text value dictionary is horrible performance-wise because the game needs to parse the whole file to create the dictionary.

5.2.3 How this localization system works

This localization system uses a simple plain text file for each language. When the user changes languages, the game builds a dictionary of key-text pairs using the language-specific file. These files are considerably smaller and quicker to read for the P.C., and adding new language support is as simple as dragging a new file into the project. The system has three main scripts:

- **TXTLoader.cs**: a simple Csharp class that loads the specific language file from the Resources folder, parses it and builds the dictionary.
- **LocalizationSystem.cs**: A singleton that handles the language selection and stores the key-value pair dictionary.
- **TextLocalizer.cs**: a MonoBehaviour that updates the string of a text object based on its key.

When the TXTLoader finishes parsing the .txt file and builds the dictionary, the game throws a Csharp delegate Action event named LanguageChanged that notifies all objects with the TextLocalizer component that they should update the text values.

If the player still hasn't set a Language in the settings menu, the system will automatically set the language to the device's language (defaulting to English). If the player has set a language, the system stores the language in the PlayerPrefs class of Unity. (The PlayerPrefs class allows the developer to store non-sensitive data easily)

5.3 Tooltip System

When designing an interface, developers face the highly complex challenge of presenting the right amount of information and context to the player without overwhelming them

with walls of text. One tool that developers have at their disposal to create visually appealing interfaces is icons. While icons are much easier on the eye, what is won in readability and aesthetics is usually lost in information. This is why most software applications have tooltips. Tooltips are brief informative messages that appear when a user hovers or interacts with a U.I. element, such as an icon. Tooltips are handy for conveying extra information to the player as they interact with parts of a game without disrupting the gameplay and keeping the interface clean.

5.3.1 Objectives

In a similar vein to the Localization System, the objective for this project was to create a future-proof tooltip system that would be usable in forthcoming projects. The goal was to make a system that was:

- **Simple**: Adding new tooltips is as simple as dragging and dropping a component to a given Game Object.
- **Future-proof**: Adding this system in future projects is as simple as importing a Unity package.
- **Visually appealing**: Tooltips flow with the game. Their presentation is coherent, clean, and not disruptive.
- **Clear and concise:** The information shown by the tooltips is appropriately formatted. Tooltips resize dynamically and change their position based on the pointer position.

5.3.2 How the Tooltip System works

The system works with three main scripts: TooltipSystem.cs, TooltipTrigger.cs and Tooltip. Each class plays a crucial role in the system's success.

The main class is TooltipSystem, a singleton class component of a Canvas Game object. This canvas has a child Game Object with the Tooltip component attached. The TooltipSystem has access to this child Tooltip object.

The singleton has public methods to show, hide, and update the Tooltip. When the pointer (mouse/finger) hovers over a Game Object with the TooltipTrigger component (that stores the tooltip information), the trigger calls the Update and Show methods of the TooltipSystem. When the pointer exits, it calls the Hide method.

The most complex class is, by far, the Tooltip class. This class resizes the Tooltip dynamically to achieve the desired behavior based on the current text values, screen size, and other parameters. To consistently achieve this, the only solution found was to play with the Layout components provided by Unity. These components leave much to

be desired and behave unintuitively, making working with them a Sisyphean task. After some trial and error, the Tooltip behaved as intended by turning on and off specific components based on the current number of characters that had to be displayed. The Tooltip also supports the inclusion of a header with bold capitalized letters. The Tooltip resizing takes into consideration the presence of the header. Finally, the Tooltip anchor changes based on mouse position to ensure the information does not show offscreen.

5.4 New Input System

Unity released the New Input System package in 2019. The old system (called Input Manager) was designed before Unity supported the many platforms that it does nowadays. As time passed, some glaring flaws of the Input Manager started to show, and a new system was needed. A new Input System was designed to work with multiple controllers and devices, be consistent across different platforms, and focus on ease of use and flexibility.

Despite considerable experience with Unity, the New Input System has never been used in the over a dozen Unity projects developed or participated in. As mentioned several times during this report, this project serves as a learning journey to advance a career in game development, providing an ideal opportunity to explore and understand the New Input System.

5.4.1 New Input System versus Old Input Manager

The old Input Manager is still a perfectly usable system and was a valid solution for the aforementioned projects. It is straightforward and fast to implement, allowing developers to prototype quickly. To gather input, a simple if statement in any Update() method is needed.

The main selling point of the New Input System is cross-platform compatibility, something that none of those projects needed. Setting up the controls in the new system is more time-consuming, has a steeper entry barrier and learning curve, and requires you to adapt your workflow. The New Input System and the Input Manager are not mutually exclusive. Both can coexist in a project and have their use cases, although it is discouraged.

The new system works with an Input Action asset, a table of possible Actions, and the inputs that trigger those actions. For each action, developers have to make Action Maps. For example, for the action "Moving," the developers could implement an action map that maps WASD to movement and another that maps a Joystick movement to movement. The action will be triggered when any input of the action maps is detected. This makes adding controller support to a P.C. game just a few clicks away and makes

it super easy to add the option to remap controls for the players.

5.4.2 Implementing the New Input System to the project

As stated before, the new system's entry barrier is steeper. It took some time to get things working and understand how the system worked and what an appropriate workflow was. After many tutorials, all mouse functionalities were correctly implemented, but that was not the case for implementing touch controls.

Unity has an in-built device simulator to test different touch devices. This is extremely handy for testing mobile games and checking that the U.I. resizes correctly. Sadly, the Unity version (2022.3.22f1) had a significant bug with the device simulator. The finger's position always returned (-infinity, infinity) when using the New Input System touch controls on the simulator. This bug was highly disruptive, and many hours were spent trying to debug the issue. It became clear that it had been an Editor bug after exporting to a smartphone and seeing everything work correctly. Luckily, Unity provides a way to simulate touches with the mouse on the default game window without needing the simulator.

Surprisingly, the New Input System lacks options that are present in the old Input Manager. Features such as registering if an object has been clicked (OnMouseDown event on the old Input Manager) are not present and must be coded. The old Input Manager also directly translated all mouse events to touch and translated tapping the screen as left-clicking. That feature is also not present in the New Input System.

For Android apps, it is recommended that users quit by pressing the "back arrow" at the bottom of the screen. The old Input Manager translated pressing that key as pressing "Escape" on a keyboard and even had an option called "backButtonLeavesApp." None of those features are present in the New Input system either, even if users have requested them in the Unity Forums.

This raises questions about whether the New Input System is more effective than the Old Input Manager, particularly in the context of this project. It also highlights the challenge developers face in not combining both systems, given that each lacks certain features found in the other. More insights about the dichotomy of both systems can be found in the section 5.7 *Results*.

5.5 Trophy System

Many of the features developed for this project differentiate a demo from a finished game, and trophies are no exception. Trophies in video games serve as powerful tools for player engagement, creating a structured system of goals that can extend a game's

longevity and replayability. The idea of game achievements can be traced back to 1982 [15] and nowadays, it is almost unthinkable for a game to be released on either console or P.C. without achievements/trophies.

5.5.1 How the Trophy System works

Designing a Trophy System is conceptually simple but implies data persistence, which is always tricky. Essentially, developers have to declare a Trophy class with a series of attributes (in this case, name, icon, and description) and a way to track if a trophy has been achieved. In A Better Tomorrow, Trophies have a boolean *isAchieved* set to true when a determined event triggers. When the user quits the game, the application loops through the list that stores all the trophies and generates a list with those that have *isAchieved* as true. This list is then converted into a JSON string stored on the Player Prefs class. When the application boots, it retrieves this stored JSON string and rebuilds the lists of achieved trophies. Then, in the list containing all the trophies, every trophy found in the achieved trophies list is marked as achieved.

All the standard trophies in the current version of A Better Tomorrow can be found listed in Table 5.2

Id	**Name**	**Description**
00	New Life	Plant your very first sapling
01	Green energy apprentice	Build your first green energy generator
02	Dam well!	Build your first dam
03	Too hot! Isn't it?	Build your first gas plant
04	Power revolution	Build your first nuclear plant
05	Strategist	Build a generator that benefits from the current weather
06	Poorly planned	Start a day with 0 generators benefiting from the weather
07	Renewables Expert	Generate 10.000 energy only using renewables in a game
08	Energy overflow	Generate 50.000 total energy
09	Power to the people!	Provide energy to a village or city
10	Provider	Provide 10.000 energy to cities and villages
11	A Better Tomorrow	Complete the game
12	The colors of nature	Unlock every theme
13	Heatproof	Overcome the Sweaty Summer Week challenge
14	Heavy blanket	Overcome the Freezing Winter Nights challenge
15	Green energy connoisseur	Last for 10 days in endless mode

Table 5.2: A Better Tomorrow standard trophies

5.5.2 Final touches

A trophy expositor was added to the main menu, allowing players to scroll through the list of all the trophies and see those that have been achieved and those that haven't. This further increases player engagement, providing goals to the players and a way to show others their progress.

Additionally, four hidden trophies were added to the game. These Easter Eggs[17] serve no purpose other than the extra enjoyment added by discovering them.

All the hidden trophies in the current version of A Better Tomorrow can be found listed in Table 5.3

Id	Name	Description
16	Orange blossom coast	Experience sunny weather three consecutive days
17	Fluffiest Friend	Find a cat
18	Don Quixote	Defeat two giants (remove two windmills)
19	That was not supposed to happen	Finish a game with 0 trees.

Table 5.3: A Better Tomorrow hidden trophies

A Better Tomorrow currently has 20 trophies in total.

5.6 Graphics

As stated in the *Planning chapter*, game art has arisen as a particular weakness during the development of this video game. Possessing some skills using Illustrator, vector graphics were chosen for their mathematical precision, enabling the creation of decent-looking imagery.

Every game needs a logo. A logo is a recognizable mark that distinguishes a game from others, a symbol that represents the game. Logos have various use cases, like promotional material, splash screens, main titles, etc. After some iterations, this is the definitive logo of the game.

Figure 5.1: Game logo.

The logo is surrounded by a seamless pattern composed of in-game sprites. The pattern includes the two most iconic green energy generators (windmill and solar panel), some trees, and some houses. Inspired by games like Mario Kart 8, this composition will be used in loading screens.

The biggest challenge was creating the vegetation. Creating good-looking natural shapes can be tricky. After some trial and error, these were the definitive trees that made it into the game (see Figure 5.2). The tops and trunks are separated to allow for procedural tree generation, and the tops are on gray scale so they can be palette-swapped. With simple scripts, each tree can have a different trunk, top, size, and color (see Figure 5.3).

Figure 5.2: Tree sprites.

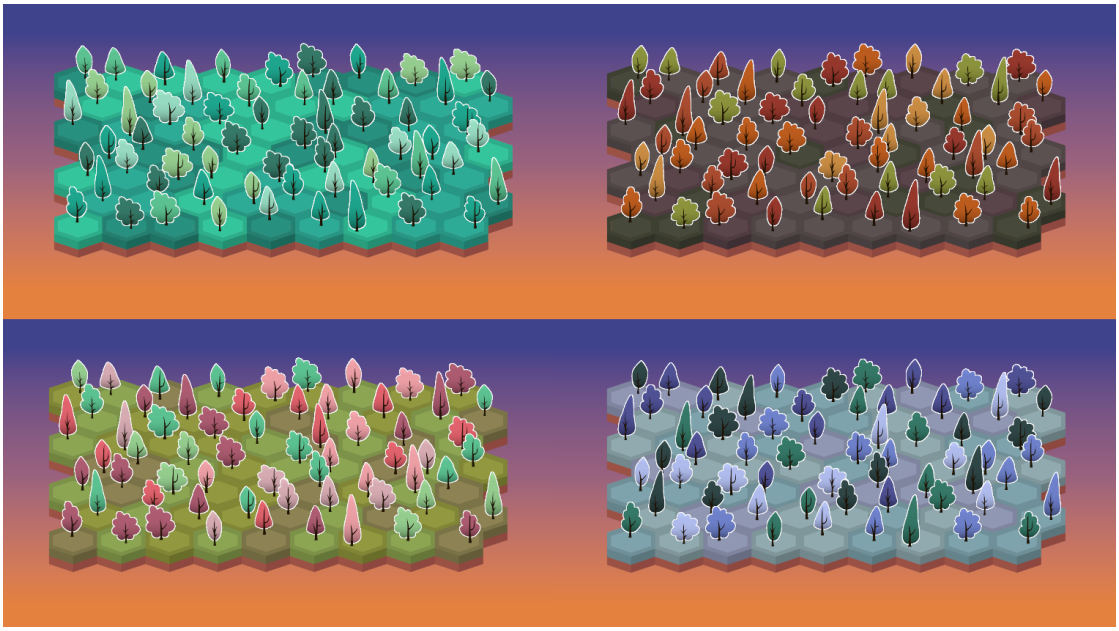

Figure 5.3: Different tree palettes, based on the selected theme (from left to right, bot to top: Meadow theme, Autumn theme, Cherry Blossom theme, Winter theme)

The remaining structures were easier to make. After all, human-made shapes usually follow straighter lines and angles (called "Hard surfaces" in 3D modeling). Villages and cities work similarly to trees to allow some variation. Facades and rooftops can be changed in color, size, and appearance programmatically. (see Figure 5.4 and Figure 5.5)

Figure 5.4: In-game structures. The red square is a placeholder for the gas plant.

Some extra details were added, leaning into the cozy aspect of the project. For example, during nighttime, and if it is not raining, there is a small chance for a cat to

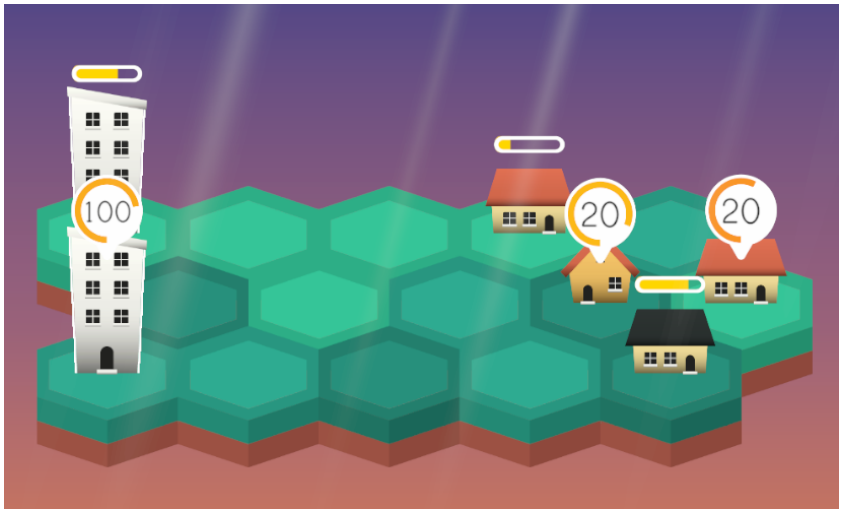

Figure 5.5: In-game towns. Currently, there are two village types and one city.

appear on a house's roof. During the day, if the weather is sunny, there is a small chance for a bird flock to cross the screen.

Some effects were added using All 1 Sprite Shader, a Unity asset that allows developers to add simple effects to sprites. An attempt to create custom shaders for sprites, like a wind effect, was made, but it was too time-consuming. Buying this asset drastically sped up the creation of visual effects, although it took some time to understand how to use it well to get the desired results.

All interface icons and shapes were also made in Illustrator. See section 3.4 *Interface Design*)

5.7 Results and conclusions

5.7.1 Localization System results

Developing a localization solution was straightforward and agile. It was expected to invest 25 to 30 hours of work into its development, but it took significantly less time to meet all the proposed goals. It was a unique and enriching learning experience. This system has already been imported to another project and worked without any issue.

This localization system will be released as CC0 (Creative Commons 0, no copyright) for everyone.

5.7.2 Tooltip System results

Developing this system was way more complicated than expected. U.I. work is a beast of its own in game development, and making interfaces that fulfill their purpose of providing enough information to the player while not cluttering the screen is extremely hard. Despite this, the developed system meets the proposed objectives.

Unity has acknowledged that the current U.I. workflow and components are subpar compared with the rest of the engine's features and has recently launched a new package: U.I. Toolkit. U.I. Toolkit allows developers to code U.I. in a style that is closer to how web pages (CSS) work. It is meant to be the standard for future engine versions (Unity 6 and onwards) but is currently in beta.

This makes one of this system's goals (future-proof) somewhat shakey, but as long as the old U.I. workflow is supported, this system will be usable.

This system will also be released as CC0 for everyone to use.

5.7.3 New Input System results

After implementing the new system for the first time, it is acknowledged to be robust and has superior capabilities. The knowledge gained through this experience is greatly appreciated and will undoubtedly be applied to future projects.

However, it is regretted taking the time to do so for this project. It was an extremely time-consuming task for little to no benefit in the context of this game. This system is undoubtedly designed to make other types of games that need to support a variety of controllers. A Better Tomorrow is (almost) an idle game. The video game can be played just with the mouse, and as stated before, the old Input Manager translates mouse input to touch input by default. It is true that the new Input System also offers an Enhanced Touch class for more complex behaviors, but once again, no complex functionality was needed to handle this game's input.

The New Input System may be better, but not for this game. With the clock ticking, focusing on other features would have been more beneficial. After all, the game behaves the same as before, but close to 40 hours have been spent.

5.7.4 Trophy System results

Adding this system was relatively straightforward, and the results were fantastic. The system has also been made scalable and flexible and will be used on future projects. Trophies add flavor to the game and are one of the most loved features by players. The development of this project has been shared on the developer's X account (formerly

known as Twitter), and the post that showcased the trophy system gathered the most engagement from users[4].

Chapter 6

Conclusions and Future Work

Contents

In this chapter, the conclusions of the work, as well as its future extensions, are shown.

6.1 Conclusions

Game development is a slow process. Most games have multi-year development cycles and are made by teams of several talented individuals. Developing a video game is hard. Developing a video game alone is harder. Developing a video game alone and in less than 300 hours (if you strictly follow the subjects' schedule) is almost impossible.

It's not uncommon for bachelor's theses to be left as unfinished projects, such as demos, vertical slices, prototypes, or proof of concepts. However, I was determined to make a mark and conclude my university journey with a fully realized and published game.

Choosing an existing project as a base was a great decision. It gave me a crystal-clear vision of the project and allowed me to jump straight into development. I could not have made it in time if I started from scratch.

There is a massive difference between a game in beta state and a so-called "Gold Master" (a version ready to be released to the public). The Devil is in the details. All the "little things" accumulate into an unimaginable amount of work. I remember hearing a professional developer say, "The last 10% of a game is as exhausting as the 90% that came before." in a GDC talk. I could not agree more.

There are countless stories online of aspiring game developers with hard drives filled with unfinished projects and ideas. Finishing is a crucial skill, sometimes unjustifiedly ignored. If we, as game developers, just prototype, we would be missing a significant part of what game development is about.

Finish. Publish. Learn. Grow.

6.2 Future Work

I conceive this project as completed. As stated in the Introduction section, a key objective of this project was to deliver a feature-complete product ready for publication in a store and take the opportunity to learn the publishing process of a video game in an store like Google Play.

I am considering publishing the game on Steam for free. This project is not suited to compete in the hostile Steam market of premium PC games, but I am interested in learning about Steam's publishing process. However, The 100 euro fee to publish a game is discouraging.

Bibliography

[1] Adobe. Illustrator. https://www.adobe.com/products/illustrator.html.

[2] Atlassian. Trello. https://trello.com/home.

[3] Universidad Europea. Developer salaries. https://creativecampus.universidadeuropea.com/blog/cuanto-gana-disenador-videojuegos/#:~:text=El%20salario%20medio%20de%20un,90.000%20euros%20brutos%20al%20a%C3%B1o.

[4] GoragarX GameDev. Trophy system post. https://twitter.com/GoragarXGameDev/status/1788495691683176534.

[5] PC Gamer. Cozy games trend. https://www.pcgamer.com/hang-on-because-the-cozy-games-trend-hasnt-even-peaked-yet/.

[6] GitHub. Github desktop. https://desktop.github.com/.

[7] Idealista. Office rent. https://www.idealista.com/alquiler-oficinas/castellon-de-la-plana-castello-de-la-plana-castellon/. Visited: May 2024.

[8] Grammarly Inc. Grammarly. https://www.grammarly.com/about.

[9] Toukana Interactive. Dorfromantik steam page. https://store.steampowered.com/app/1455840/Dorfromantik/.

[10] JetBrains. Jetbrains rider. https://www.jetbrains.com/rider/.

[11] Odin. Odin inspector. https://odininspector.com/.

[12] Overleaf. Overleaf. https://es.overleaf.com.

[13] Seaside Studios. All in 1 shader. https://assetstore.unity.com/packages/vfx/shaders/all-in-1-sprite-shader-156513.

[14] Unity Technologies. Unity3d. https://unity.com/.

[15] Wikipedia. Achievements in video games. https://en.wikipedia.org/wiki/Achievement_(video_games).

[16] Wikipedia. Asynchronous c.a. https://en.wikipedia.org/wiki/Asynchronous_cellular_automaton.

[17] Wikipedia. Easter eggs. https://en.wikipedia.org/wiki/Easter_egg_(media).

[18] Wikipedia. Moore neighbourhood. https://en.wikipedia.org/wiki/Moore_neighborhood.

[19] Todo Luz y Gas. Cuanto gasta una empresa en luz al mes. https://www.todoluzygas.es/blog/luz/cuanto-gasta-una-empresa-en-luz-al-mes#:~:text=El%20gasto%20mensual%20de%20luz,o%20el%20n%C3%BAmero%20de%20empleados.

Appendix A

Source code

A.1 Project GitHub

All the project files can be found *here* (https://github.com/GoragarXGameDev/VJ1241)

A.2 Cellular Automata

```
using UnityEngine;

// +----------------------------+
// | ---- GoragarX GameDev ---- |
// | goragarxgamedev@gmail.com  |
// +----------------------------+

[System.Serializable]
public class CellularAutomatonData
{
    /// <summary>
    /// Auxiliary class to set the Cellular Automaton data in Unity's inspector.
    /// </summary>

    public TileBase[] cellStates;
    public bool useRandomSeed;
    public string seed;
    [Range(0,100)] public int fillPercentage;
    public int generations;
}

public class CellularAutomaton
{
```

```
24      /// <summary>
25      /// Basic cellular automaton.
26      /// This C.A. uses Moore Neighbourhood and fully-asynchronous cell state updates.
27      /// You can modify the initial state and evolution rule(s) as desired.
28      /// The following is just an example.
29      /// </summary>
30
31      public int[,] CellGrid { get; private set; }
32
33      private readonly int _x, _y;
34      private readonly CellularAutomatonData _data;
35
36      public CellularAutomaton(int x, int y, CellularAutomatonData data)
37      {
38          _x = x;
39          _y = y;
40          _data = data;
41      }
42
43      public void GenerateCellGrid()
44      {
45          CellGrid = new int[_x, _y];
46
47          SetInitialState(); // You can change the Initial State by modifying this method.
48
49          for (int i = 0; i < _data.generations ; i++)
50          {
51              UpdateCellsStates(); // You can change the evolution rules by modifying this method.
52          }
53      }
54
55      /// <summary>
56      /// Randomly fills the grid, setting the state of each cell to 0 or 1.
57      /// The percentage of 0 or 1 depends on CellularAutomatonData.fillPercentage;
58      /// </summary>
59      private void SetInitialState()
60      {
61          if (_data.useRandomSeed) _data.seed = Time.time.ToString();
62          System.Random pseudoRandom = new System.Random(_data.seed.GetHashCode());
63
64          for (int x = 0; x < _x; x++)
65          {
66              for (int y = 0; y < _y; y++)
67              {
68                  // If map border (border width = 2), set value to 0.
69                  if (x <= 1 || x >= _x - 2 || y <= 1 || y >= _y - 2)
70                  {
71                      CellGrid[x, y] = 0;
72                  }
73                  else // else, set randomly.
74                  {
75                      CellGrid[x, y] = pseudoRandom.Next(0, 100) < _data.fillPercentage ? 1 : 0;
76                  }
77              }
```

```
78          }
79      }
80
81      /// <summary>
82      /// Evolution rule: each cell state changes based on its neighbours' state.
83      /// Fully-asynchronous update: each tile is updated on evaluation.
84      /// </summary>
85      private void UpdateCellsStates()
86      {
87          // For a synchronous update, create a copy of the CellGrid and update the states there.
88          // When all cells have been evaluated, set the copy as the CellGrid.
89
90          for (int x = 1; x < _x - 1; x++)
91          {
92              for (int y = 1; y < _y -1 ; y++)
93              {
94                  // Get each tile neighbours (each tile has a maximum of 8)
95                  int neighbourCellsState = GetNeighboursStateSum(x, y);
96
97                  // If it has more 1 than 0, set to 1.
98                  if (neighbourCellsState > 4) CellGrid[x, y] = 1;
99
100                 // If it has more 0 than 1, set to 0.
101                 else if (neighbourCellsState < 4) CellGrid[x, y] = 0;
102
103                 // If it has the same amount, do nothing.
104             }
105         }
106     }
107
108     /// <param name="gridX">grid x coordinate</param>
109     /// <param name="gridY">grid y coordinate</param>
110     /// <returns>The sum of the neighbour cells' state values,
111     /// given a cell coordinates</returns>
112     private int GetNeighboursStateSum(int gridX, int gridY)
113     {
114         int count = 0;
115
116         // Using Moore neighbourhood: the 8 surrounding cells of a given cell
117         // (NW, N, NE, W, E, SW, S, SE).
118         for (int neighbourX = gridX - 1; neighbourX <= gridX + 1; neighbourX++)
119         {
120             for (int neighbourY = gridY - 1; neighbourY <= gridY + 1; neighbourY++)
121             {
122                 // To avoid IndexOutOfRangeException
123                 if (neighbourX >= 0 && neighbourX < _x && neighbourY >= 0 && neighbourY < _y)
124                 {
125                     // Don't evaluate the cell itself
126                     if (neighbourX != gridX || neighbourY != gridY)
127                     {
128                         count += CellGrid[neighbourX, neighbourY];
129                     }
130                 }
131             }
```

```
132         }
133
134         return count;
135     }
136 }
```