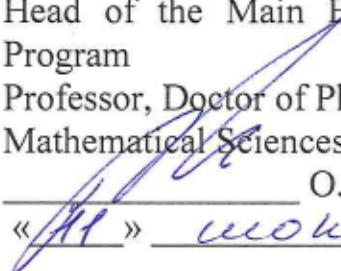


Ministry of Science and Higher Education of the Russian Federation  
NATIONAL RESEARCH  
TOMSK STATE UNIVERSITY (NR TSU)  
Research and Education Center "Higher IT School" (HITs)

APPROVED BY  
Head of the Main Educational  
Program  
Professor, Doctor of Physical and  
Mathematical Sciences

  
O.A. Zmeev  
« 11 » 06 2021

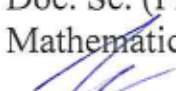
**BACHELOR'S THESIS**

**CREATING 2D GAME "LITTLE SIM WORLD" USING UNITY ENGINE**


Main Educational Program  
02.03.02 – Fundamental Computer Science and Information Technology  
Specialization "Software Engineering"

Neffati Housseem

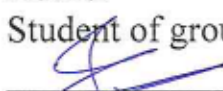
Bachelor's Thesis Supervisor  
Doc. Sc. (Physics &  
Mathematics), Professor

  
A.V. Kitaeva  
« 11 » 06 2021

Consultant  
The head of Digital Service  
Department of TSU

  
D.A. Sokolov  
« 09 » 06 2021

Author

Student of group No 97A05  
  
H. Neffati  
« 09 » 06 2021

The Ministry of Science and Higher Education of the Russian Federation  
NATIONAL RESEARCH TOMSK STATE UNIVERSITY (NR TSU)

REC "Higher IT School"

APPROVED  
MAP Head  
Dr. Sc. (Phys.-Math.), professor  
O.A. Zmeev  
27.02 2021

THE TASK

Regarding the bachelor's final qualification work implementation by a student  
**Neffati Houssem**

(student's full name)

In the field of study Fundamental Computer Science and Information Technology, specialization  
"Software Engineering"

**1. The bachelor's final qualification work theme** (in the English and in the Russian language)

**Creating 2D game "LITTLE SIM WORLD" using Unity Engine / Создание 2D игры "LITTLE  
SIM WORLD" с использованием Unity Engine**

**2. The deadlines for the completion of task:**

a) to academic office – 04.06 2021.

б) to State Examination Board – 11.06 2021.

**3. Description:**

The purpose of this work is to develop a 2D life simulator game that brings competition to its

goals and objectives of the FQW, expected results

genre. The expected results: to create the base mechanics and an extensive content for the

game; to create and deploy in the game several systems that are the base of this game genre

such as time, weather, population, buildings, character creation and animations.

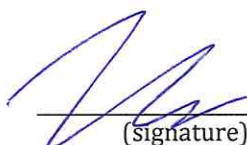
**The organization that is being researched:**

Tomsk State University, Digital Service Department

**Final qualification work supervisor**

Doc. Sc. (Physics & Mathematics),  
Professor of Software Engineering  
Department

(position, place of work)

  
(signature)

/ A.V. Kitaeva  
(initials, surname)

**Final qualification work consultant**

The head of Digital Service Department  
of TSU

(position, place of work)

  
(signature)

/ D.A. Sokolov  
(initials, surname)

The task is accepted for execution

27.02 2021  
(date)

  
(signature)

/ N.Houssem  
(initials, surname)

## **ABSTRACT**

Bachelor's thesis 46 pages, 47 figures, 8 sources, 1 appendix

**The purpose** of this work is to develop a 2D life simulator game that brings competition to its genre.

**Work results:** The base mechanics and an extensive content for the game have been created.

Several systems that are the base of this game genre such as time, weather, population, buildings, character creation and animations have been created and deployed in the game.

Research methods are theoretical research and practical implementation.

**Key words:** Unity, Video Game, 2D, Life Simulator

## CONTENT

Glossary	3
Introduction	5
1 Software theory of creating video games	7
1.1 Overview of general classifications of video games	7
1.1.1 Classification of games by platform	7
1.1.2 Classification of games by graphics	8
1.2 Analysis of game engines	10
2 Requirement analysis, development tools and high-level system presentation	12
2.1 Functional and non-functional requirements	12
2.2 Problem domain model	13
2.3 Domain formalization	13
2.4 Technologies and tools	17
3 Game engine and tools overview	21
3.1 Unity Engine	21
3.2 Unity Animation.	22
3.3 Unity Sprite Editor	23
3.4 Unity Tilemap Palette	25
4 Technical game analysis	26
4.1 Overview	26
4.2 Systems	26
4.3 Mechanics	36
4.4 Flows	38
Conclusion	41
References	42
Appendice A Additional illustrations	43

## GLOSSARY

**Unity:** The game engine that will be used to develop this project.

**Steam:** A platform where developers can publish their games and people can buy and download their games on.

**Life Simulator Game:** A game in which the player lives or controls one or more virtual characters (human or otherwise). Such a game can revolve around individuals and relationships, or it could be a simulation of an ecosystem.

**2D Game:** Everything in the game is happening in a 2D plane.

**NPC (Non-Player Character):** Characters in the game that are not controlled by human players.

**Game object:** Anything in the game that the player can see.

**Particle system:** Technique used in games that allows to reproduce otherwise difficult phenomena's like fire by using a lot of small sprites.

**Playability:** A measure that indicates if the game can be played or not. Not just in the sense of controlling the game, but also taking into consideration the quality of it.

**Published:** Put the game in a store to be sold.

**Scene:** A file in which each level is stored and contains the objects and characters that exist in the game world.

**Sprite:** Sprites are simple 2D objects that have graphical images (called textures) on them. Unity uses sprites by default when the engine is in 2D mode.

**Tilemap sprite sheet:** A set of sprites that are used to create a map.

**Game save:** A file written in the local disk that contains all the data and progress of the player in the game.

**Interactable object:** Any object in the game that the player can interact with, such as furniture, NPCs, shops, cars, etc..

**Graphic settings:** Menu of options that can be changed to adjust the quality of the game's display and level of details.

**Bug:** An error in the game that leads to a wrong behavior of something or anything that is not how it should be E.g.: a character going through a wall.

**Multiplayer:** A game mode that allows multiple players to play together in the same game world while connected to the same server.

**Cheats:** Software or tricks to change the original behavior of the game to give unfair advantage to the player.

**Buff:** In gaming, a buff is an effect placed on a character that enhances their statistics or characteristics. buffs may be applied through gameplay, may be bought, or may be applied by game developers. BUFF is an antonym of DEBUFF (Harmful Effect)

**Scriptable object:** A Unity object that has a C# script attached to it with preset variable configuration.

**XP (Experience Points):** Points that the player character receives to represent his professional or personal growth in the game world.

**UI (User Interface):** This is the point of human-computer interaction and communication in a device. This can include display screens, keyboards, a mouse and the appearance of a desktop. It is also the way through which a user interacts with an application or a website.

# **INTRODUCTION**

## **Background**

This work is written on the basis of a development project of a 2D life simulator game. In order to comply with the non-disclosure agreements, only data that is non-sensitive and public will be used for this work.

The primary goal of this project is to develop an alpha version of the game with core mechanisms and features.

## **Motivation**

Noticing that “LITTLE SIM WORLD” game project doesn’t have any competition in the market and is missing a lot of features and improvements that fans have been asking and wishing for, for years, the motivation is to start a project to bring something new to the market, something that will satisfy these fans’ wants and requests.

## **Problem formulation**

Using the knowledge learned during this bachelor degree and work experience in order to develop and design a prototype of a videogame using the Unity engine. Parts of the project that are not directly related to the architecture or code are not a part of this paper. The art of the game is made by a paid artist as my objectives and motivations are not related with art.

## **Specific objectives**

- Design from scratch the whole game including writing the game’s scripts, creating the game character models and their animations and the game world scene.
- Learn to manage tasks and carry out the project satisfactorily.
- Achieve a fun gameplay for players by playtesting what will be made.
- Learn to balance the game in order to make it fun to play.
- Implement all base life simulator game components at the highest levels of industry standards.

## **Scope of the project**

The final scope of the project is to release a game that is accepted by the target audience of this project and at the level of expectations.



# **1 SOFTWARE THEORY OF CREATING VIDEO GAMES**

## **1.1 Overview of general classifications of video games**

A computer game is a form of developmental and entertaining interaction between a user and a computer, imitating life and imaginary situations in virtual space, having a significant educational potential, which is to stimulate cognitive interest.

There are a lot of computer games created, and from year to year there are more and more of them. Computer games can be divided into groups, using various methods.

The main way to categorize video games is by platform, which indicates which device you can run them on. If the user does not have a platform for which the game is intended, then he will not be able to play it.

### **Personal computer (PC, laptop, netbook)**

Computers are the main platform for video games. The original job of computers was to perform complex scientific calculations, but later they also took on the role of the electronic typewriter, and then the role of the gaming platform.

Due to its gaming focus, personal computers acquired video cards, sound cards, increased the power of processors to such heights that now they are able to display photorealistic graphics in real time.

Personal computers, in turn, are subdivided into several sub-platforms by operating systems (OS). Each operating system has its own tools for processing video games.

List of the most popular computer OS series: Windows (from Microsoft), Mac OS (from Apple), Linux (a free OS developed by the global Internet community).

### **1.1.1 Classification of games by platform**

#### **Game console (PS, Xbox, Nintendo)**

A console is a hardware equipment that is mainly created to run video games. Most consoles require a connection to a TV. But there are independent portable consoles with a built-in screen. Unlike computers, consoles are ready-made, non-

collapsible devices (only a few external parts can be replaced and updated). In this regard, the development of consoles is a process, divided into clear milestones called console generations. At the moment, the 9th generation of game consoles have already been released. The most popular generations of consoles include: Sony PlayStation (PSP, PSOne, PS2, PS3, PS4, PS5), Microsoft Xbox (Xbox, Xbox 360, Xbox One, Xbox series), Nintendo 3DS, Wii, Nintendo Switch

### **Mobile device: phone, tablet**

By technical characteristics, mobile devices are much weaker than stationary computers; therefore, mobile games look somewhat simpler than ordinary games, but the situation is gradually improving. Very often, computer games that are 5-10 years old are released on mobile phones.

On modern phones, games are run under mobile OS: Windows Mobile, Android and iOS. For mobile distribution, games have created entire global services such as the App Store and Google Play.

### **1.1.2 Classification of games by graphics**

#### **Classification by camera perspective**

the point from which we look at the game world is closely related to the genre of the game. In some genres it is easier to look from the perspective of the hero, in some it is preferable to see everything from the side, and sometimes it is convenient to observe the situation from a bird's-eye view. Some games have the ability to change the game view camera right during the gaming session.

1st person view is the view in which we see the virtual world through the eyes of the protagonist. This view is the most convenient way to get used to the role of a virtual hero. It is also well suited for aiming, therefore it is used in shooters. This view is used in these genres: action, shooter, RPG, life simulators, racing.

3D person view is the view in which we see the virtual world from the outside, so that the main character is standing in front of the screen center, allowing you to better assess the situation. The main character is always in sight, therefore his

external looks and animations should be clearly visible. This view is used in these genres: action, shooter, RPG, life simulators, racing, fighting games, 3D platformer.

2D side view is a side view that lets you see different layers of elevation of the level for example; building floors and platforms. Lack of the 3rd dimension greatly simplifies the perception of the game world. Used in genres: platformers, puzzle games, fighting games, 2D action.

### **Classification by graphics technology.**

When choosing a game, many inexperienced players are guided by the graphics, therefore there is a division of games by type and quality of the graphic image. The low graphical processing power of first generation computers caused a lot of restrictions to game developers. Because of this, many older games used text design rather than graphics. Games of this kind are more like an interactive book rather than a video game. But in our time there are similar games that are still being developed and played nonetheless. 2D graphics (vector graphics) are the most natural looking graphics. The images are composed of individual pixels or colored squares. The era of 2D games popularity passed in the 1990s, but left its mark on the gaming culture in the form of a pixelated style graphics. In the late 2000s, in the wake of the success of indie games, the fashion for 2D games returned. There is also a technology of vector images, in which objects do not consist of pixels, but of precise geometric coordinates, connected by lines. This type of image allows you to draw smoother lines without pixelation. Enlarging the image does not deteriorate its appearance.

Today 3D graphics are the most popular graphics format in computer games. Through the use of trigonometric formulas, game developers can create an illusion a three-dimensional world, displayed on a two-dimensional plane (screen). The computer calculates the real 3D models and displays the mathematically calculated 2D projections of these 3D objects. The ideas of three-dimensional images date back to the 1970s, and the real three-dimensional graphics did not appear until the early

1990s.

Augmented reality games (mobile devices with a video camera) are games where the real world is displayed through the camera lens screen, but with the addition of virtual objects. The player drives a video camera, looking for appearing objects or takes aim and destroys them. Augmented reality is an interesting and unusual idea, but it has not gained wide popularity yet.

Virtual reality (VR) implies the complete immersion of the player in the virtual world. In the current stages of game development, it is impossible to achieve full-fledged VR experience due to hardware limitations. At the moment there are VR headsets, in which are broadcast stereo graphics. Hand-held sensors can display real hand movements in virtual space. But all of these are just the first steps towards VR.

## **1.2 Analysis of game engines**

At the moment, there are a huge number of game engines. A game engine is a toolkit designed to simplify and accelerate the game development, so as not to write everything "from scratch", it is as well the module of the game that includes the graphics and display logic. The engine contains a set of systems that control certain parts of the game. For example, the engine can contain:

- Graphics subsystem (animation, object rendering, etc..)
- Sound subsystem
- System kernel (support for various platforms)
- Support for scripts and programming languages

Each engine represents a powerful game development environment. When developing games, developers sometimes create their own engines, which is more laborious and costly, but you can also use a ready-made game engine.

To develop a logical computer game with shooter elements for example, it is important that the engine has a number of characteristics:

- **Multiplatform:** The engine should allow to create games that are compatible with different platforms such as PC and consoles
- Should contain tools for working with 3D graphics and animations
- Support network mode for multiplayer gaming

**Unity Engine** is a multi-platform game engine designed for Windows, Mac OS X and Linux games. It comes with a free and PRO version and they both come with the same features and tools (PRO license required only if the company's yearly revenue is more than 100,000 USD) and Supports iOS, Android, Nintendo Wii, PlayStation 3, Playstation 4, Xbox 360 and Xbox One. All software products created in the environment of the Unity game engine have full support for the graphics technologies DirectX and OpenGL. By virtue of convenient interface, simplicity of working with the engine, as well as the availability of a free version of the game engine, Unity Engine is one of the most popular ready-made engines for developing games. The editor built into the Unity Engine has an intuitive interface that is easy to work with and adapt to. The interface consists of several windows and work panels and allows to customize the game project directly in the editor. It also has tools like the built-in terrain editor, which allows you to model game locations directly in the engine, creating complex geography and overlaying textures right in the editor. The engine supports the programming language C# by default Since version 4.0

It also comes in with a built-in animation editor Mecanim which allows to use animations with similar characters.

## **2 REQUIREMENT ANALYSIS, DEVELOPMENT TOOLS AND HIGH-LEVEL SYSTEM PRESENTATION**

### **2.1 Functional and non-functional requirements**

The requirements written below are for the final project and not the final version of the game, but nevertheless persistent throughout the whole stage of development. It is important that this game project fulfills the basic requirements of a universal video game and of this typical genre to ensure acceptance from the wide audience that we plan to target.

#### **Functional Requirements**

1. Players should be able to create and customize one or multiple characters
2. Players should be able to save and load all their game progress whenever they want
3. Players should be able to use any interactable object in the game
4. Players should be able to change the graphic options and audio/game settings anytime in the game from the options menu
5. Players should be able to pick up, remove and manage items in their inventory
6. Players should be able to speed up, slow down or pause the game
7. Players should be able to move freely around any accessible areas in the game
8. Players should be able to easily navigate between the game world scene and the main menu scene
9. Players should be able to see the current day, season, time and money in the game's on-screen UI.

#### **Non-Functional Requirements**

Stability: The game must be virtually free of bugs.

Performance: The game must be playable on Windows OS and DirectX 11.

Minimum specs required:

1. Memory: 4GB

2. GPU: Intel HD 4000
3. CPU: Intel Core 2 Duo T5600
4. OS: 64-bit Windows 7/8/8.1/10
5. Deployment: The game must be deployable on Steam.

## 2.2 Problem domain model

Life simulator games are all about “maintaining and growing a virtual life”, where players are given the power to control the lives of autonomous people or creatures. The problem presented here is such that, in order for a person to participate in an activity, he would need to have the appropriate tools and environment for that.

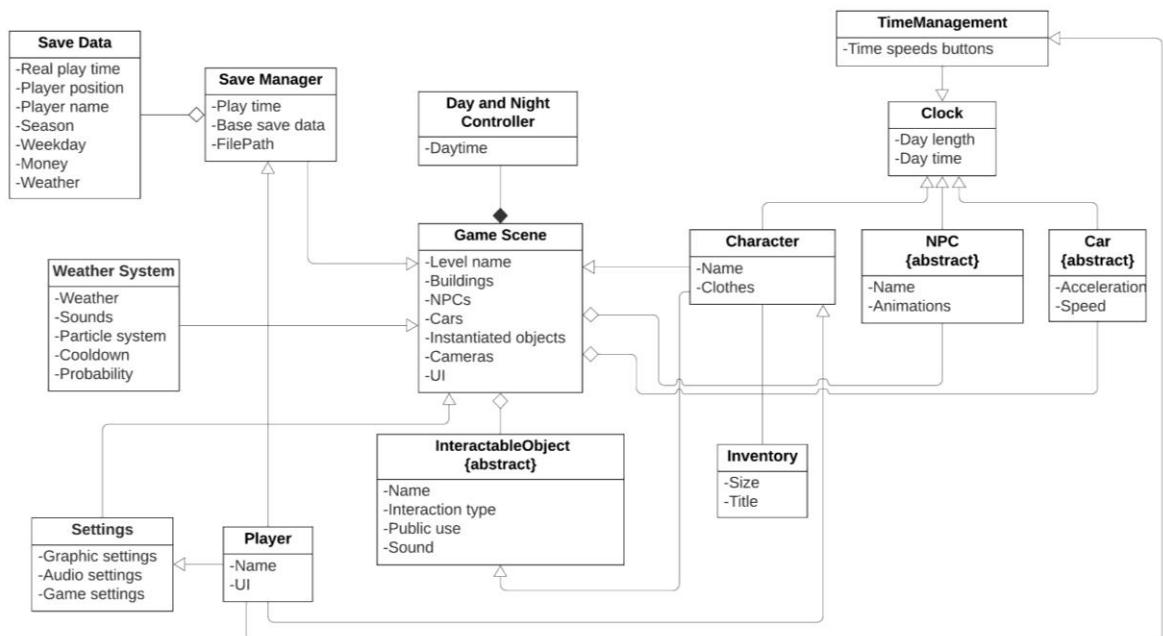


Figure 1. Class diagram, illustrating the general problem domain model

The problem here is you cannot participate in the desired activity if the environment or weather don't allow for it. For that, all essential elements illustrated in the above figure (Figure 1) need to be digitized and recreated.

## 2.3 Domain formalization

To solve the previous issue, a life simulator game offers the option to digitize a complete environment for the player to do an activity in a pre-set virtual environment. Also, other limitations or constraints such as physical movement are

not important requirements or at least, not as important as in the real-life situation.

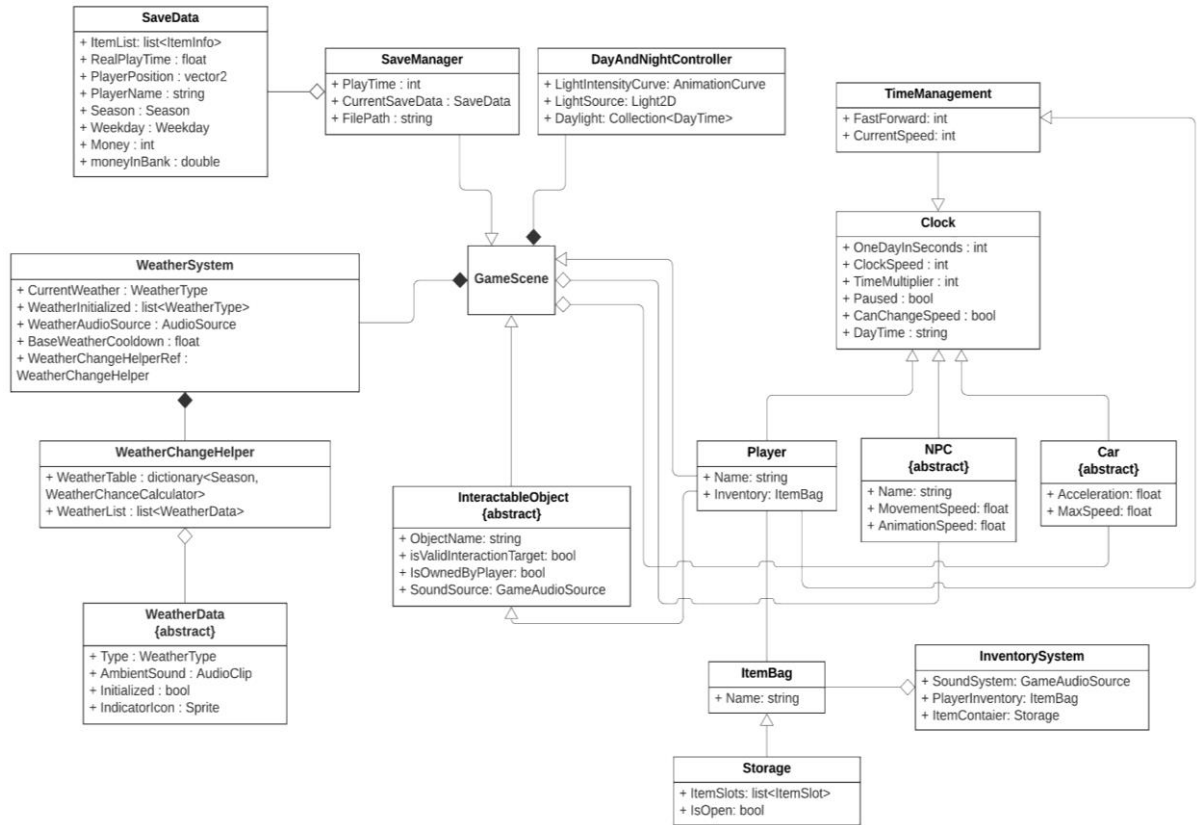
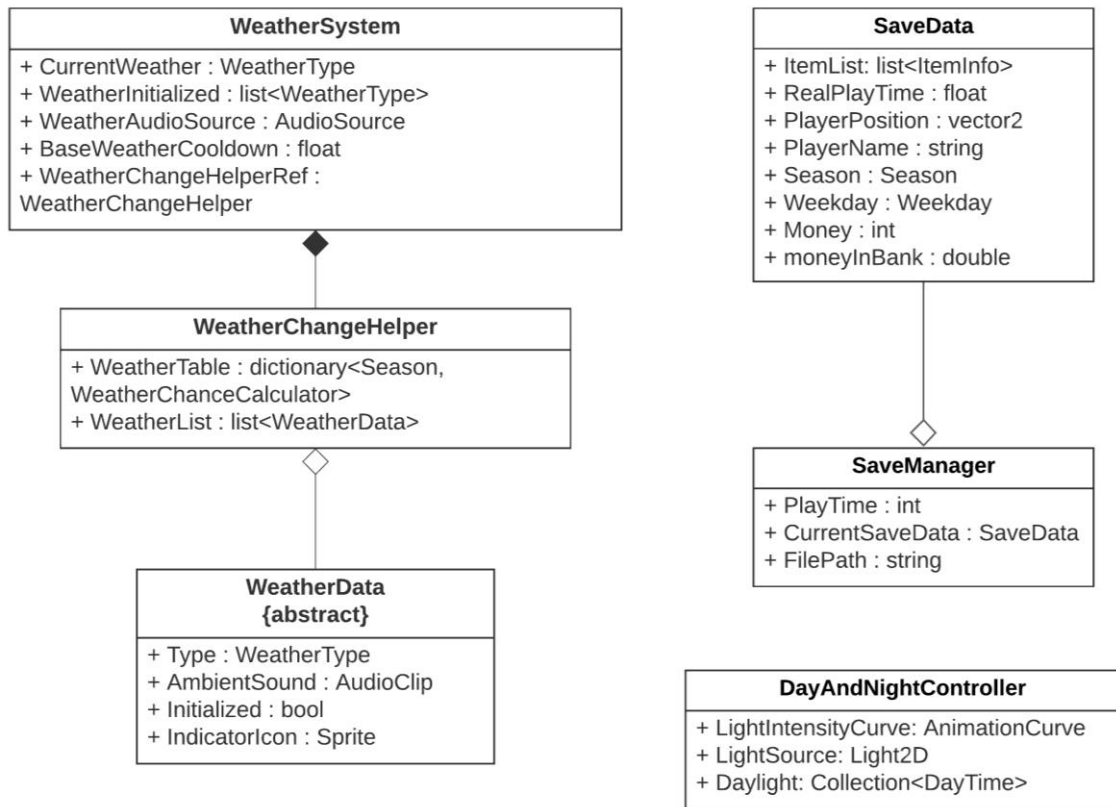


Figure 2. Class diagram, illustrating the general formalized domain model

To create a representation of a slice of real life in a virtual environment, it was decided that these elements need to exist in the game scene. These elements affect the player's experience and immersion which is what players seek in such video games.





*Figure 3. Class diagram, Clarifying major base systems*

The **Weather System** is designed to add immersion to the game and reinforce the feel of living in different seasons by casting different weathers at different seasons in the game with visual and auditory experience.

The **Day and Night Controller** is also an important part of the daily routine of the game. It controls the lights in the game, such as sunlight and buildings' lights, in a way that reflects the exact daylight in real life, but in different speed. It is also designed to change light intensity and colors dynamically based on the season of the game.

The **Save Manager** keeps most of the game's data and creates a copy every time the user saves the game or abruptly ends the game session. Data saved include the player's position and his possessions, the day cycle etc..

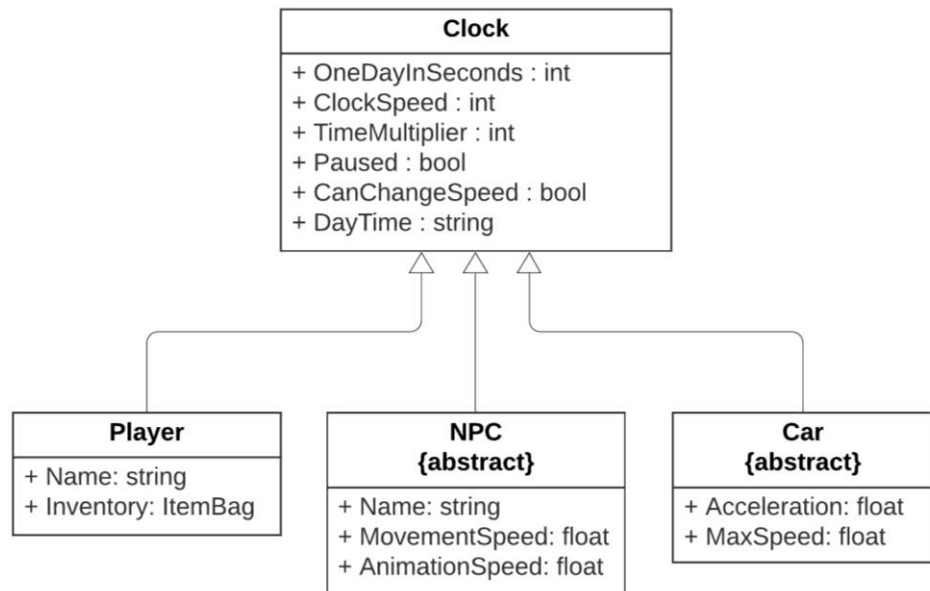


Figure 4. Class diagram, Clarifying time-governed movable object

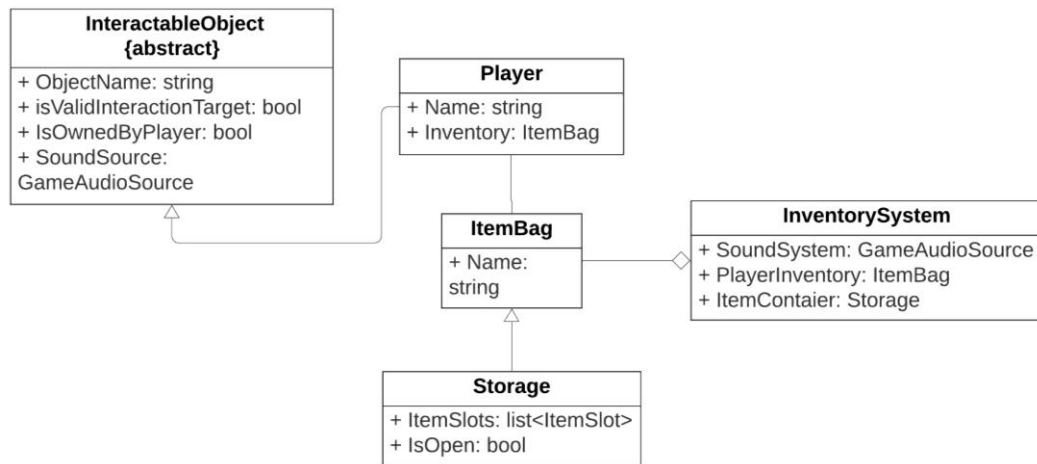


Figure 5. Class diagram, Clarifying player interaction circle

The **Clock** governs the time in the game. It controls the passing of time and the day length in seconds relative to real time and can receive instructions from the **Time Management** buttons in the player's UI to pause or speed up the game.

This system directly controls the player, NPCs and cars movement and animation speeds.

The **Inventory System** controls the interaction between the player and his inventory and other storages such as fridges or shelves. It contains the instructions on how to move items, stack them, reduce quantity and drop on the floor.

**Interactable Objects** are any object in the game that can be interacted with. It is designed to have interaction conditions and animations and play specific

sounds to increase immersion.

Some of these objects can be owned by the player while others are public.

In the game's world scene, a player can do many activities such as socializing with other players, getting a job and working or just shopping.

## **2.4 Technologies and tools**

The framework that was chosen for this project was Unity (using C# as programming language) for the following reasons:

1. Unity provides simple and intuitive UI and tools for 2D game making.
2. Unity is one of the biggest game engines in the market. It's free, stable and has long-term if not life-time technical support and updates.
3. Unity has a huge marketplace for plugins, assets and packages that are either free or paying and can be very helpful to the making of games.
4. Unity is arguably the best engine for making 2D games thanks to its Universal Render Pipeline offering the 2D renderer that perfectly handles layers and sorting orders of sprites and images as well as offers high quality graphics.
5. Unity offers internal tools such as animation tools and tile map palette which replace the paying and expensive 3rd party plugins that you would have to otherwise buy.

For project and task management, Trello was chosen because it's simple, free and very practical and fast when it comes to task assignment and bug reporting/fixing.

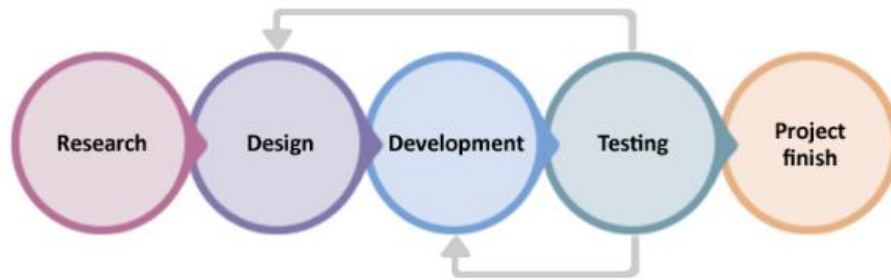
Discord is the chosen communication software for quick team chats, alerts and cooperation.

Photoshop for art creation as it is the best software for image editing and sprite drawing.

## **1.5 Methodology**

An agile methodology is a must for this project, other methodologies are way too slow and they will almost assure a failure in this project because they do not have

the fast iteration that is needed to make the game fun to play. Scrum is the most popular agile methodology in software development and specially in video games, and this is because it helps to ensure that the goals are meet with quality in each part of the development using the Sprints. For those reasons the agile methodology Scrum will be used. The following figure describes the process I will follow in a general term:



*Figure 6. Methodology (self-made)*

### **Research**

The first things to be done will be the research. A good research is indispensable, and it is the first step for the project, specially information about competence and trying to deduce what makes them good games and how to translate this to our game. After this, once the game idea is clear and it is decided what we will do, it is time to look for the art assets we want and plan all the management.

### **Design**

Once everything is planned, the design must be done following the references chosen in the research. A GDD (Game Design Document) will be made.

the game should still allow the player to access settings, but not move items in inventory. Everything should become static.

### **Day and Night cycle**

Seasons in the UK:

The number of daylight hours in-game will be decided by the Season in-game. These changes per Season are especially significant because we will, in the future, implement different kinds of weather for each Season.

Spring (March to May)

Average Min/Max Daylight hours: 13h of daylight

100% lighting from 06:30h till 19:30h.

Summer (June to August)

Average Min/Max Daylight hours: 16h of daylight

100% lighting from 05:00h till 20:00h.

Autumn (September to November)

Average Min/Max Daylight hours: 11:00h of daylight

100% lighting from 07:00h till 18:00h.

Winter (December to February)

Average Min/Max Daylight hours: 8h of daylight

100% lighting from 08:00h till 16:00h.

As far as the DAY/NIGHT LIGHTING CYCLE is concerned, when the sunlight is over, the sun will start to decrease. The dimming of light begins at the end of the daylight hours. For example, in autumn, dimming begins at 18:00. This happens over the period of 4 hours, so by 22:00, the game will be completely dark. The same is true for the sunrise, beginning 4 hours before the day starts. However, we do not have complete darkness in LSW, eg. 0 light in complete darkness. Currently we use 90% darkness so the player can still kind of see things.

Lights mechanic should be as followed:

1. The lights outside should turn ON 2h before sunset.
2. The lights outside should turn OFF 4h before sunrise.

The amount of sunlight during the day should be decreased when the weather is not sunny:

1. When the rain is falling the amount of sunlight should be REDUCED by 15% ( Value should be 85%)
2. When it is cloudy, the amount should be REDUCED by 10% ( Value should be 90%)
3. When it is sunny, the amount should be INCREASED by 5% ( Value should be 105%)

Lights in the buildings in the game shouldn't be turned on, and the value is 100% immediately. For now, let's make a gradual lightening on. Go from 0% to 100% light in houses in a period of 30 minutes. Eg. If the light outside should turn ON after the sunset (at 50% value of daylight amount), start turning it on slowly 30 minutes before that.

For future reference when we make a new weather system the references for the UK can be found on these sites:

<http://www.foreignstudents.com/guide-to-britain/british-culture/weather/seasons>

<http://www.foreignstudents.com/guide-to-britain/british-culture/weather/conditions>

<https://www.timeanddate.com/astronomy/different-types-twilight.html>

<https://www.timeanddate.com/sun/uk/london>

### **Player Camera**

The player camera cannot be shifted from the player. The player should always be at the center of the screen unless he is at the edge of the player map where collisions occur with the camera. Different zoom levels can be accessed with the mouse wheel. We will allow zoom through the settings menu in the future.

*Figure 7. Screenshot of an example page from the GDD*

The design includes the game systems, game mechanics, the UI, the player, NPCs and their behavior, the items, the music and the music effects. An art guide will also be made and will include all guidelines for artists to ensure consistency and conformity of all art in the game.

## **Development**

Right after the main parts of the GDD are finished, the game can be started. The main focus in the first stage of the development of the prototype will be creating

the main parts of the map inside Unity. Once we have that, we can start putting things on it, starting with the player, and including its skills and its level up system together with the environmental sounds. Once the player can move inside the map and the level up system is made, the NPCs will be added. Once we have both player and NPCs, it is time to start building the interaction between them and developing the AI of the NPCs with their behavior. With this we can create a spawn and behavior system. Once done, the main playability of the game will be finished.

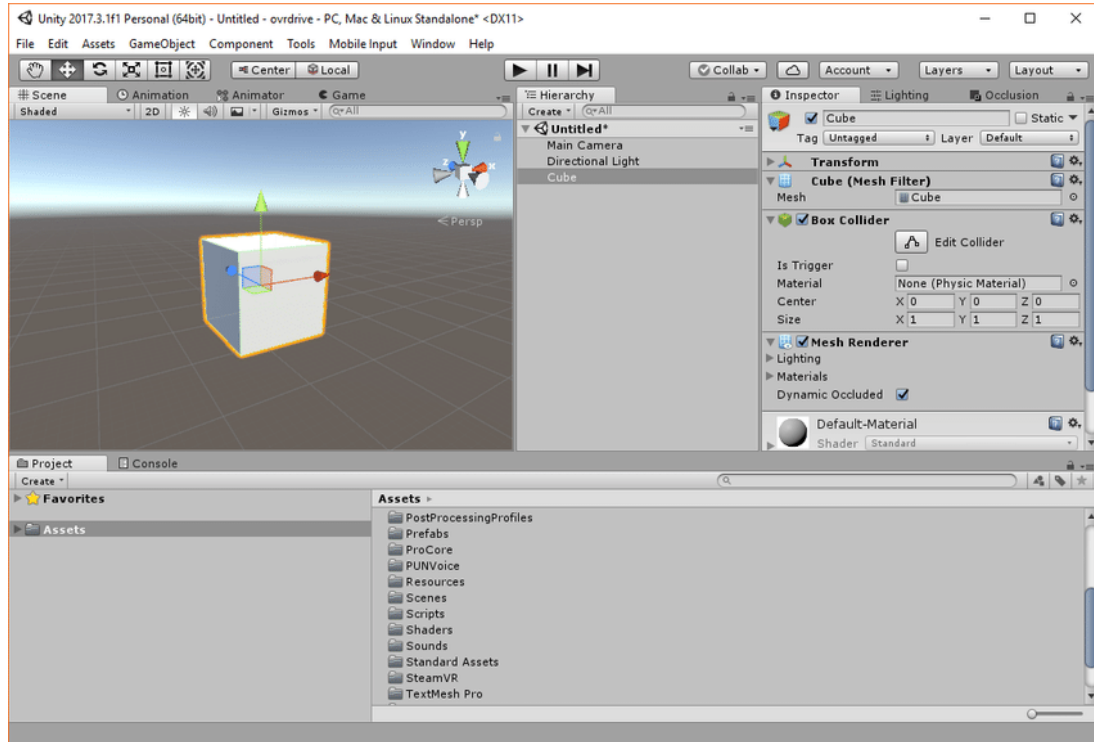
### **Testing**

This could be considered a really simplistic prototype and a simple playtest of the game can be done; however, it is too early to look for people to play the game because it would be too time consuming in exchange of little feedback. However, a QA playtesting must be done at this point. This way we can find not only bugs, but also start checking if the game is fun or not. In case of bad design or missing parts, we have to go back to design and develop some features. With this iteration we can improve the game faster and fail early in the project when we have time to adapt and solve the problems. While testing, all the final features of the prototype will be developed in parallel. This includes all game system, everything related to the UI and the players and NPCs character design. With the improvements from the iteration and with the final parts of the development finished, we can plan an external playtesting. A pre-alpha version will be released and early bird players will be giving their feedback which will be used to improve the game in future iterations.

## 3 GAME ENGINE AND TOOLS OVERVIEW

### 3.1 Unity Engine

Unity core platform enables rapid editing and iteration in your development cycles, with real-time previews of your work. You can create 2D or 3D scenes, animations or cinematics directly in the Unity Editor.



*Figure 8. Unity Editor screenshot, illustrating the main layout and menus of the Unity Engine.*

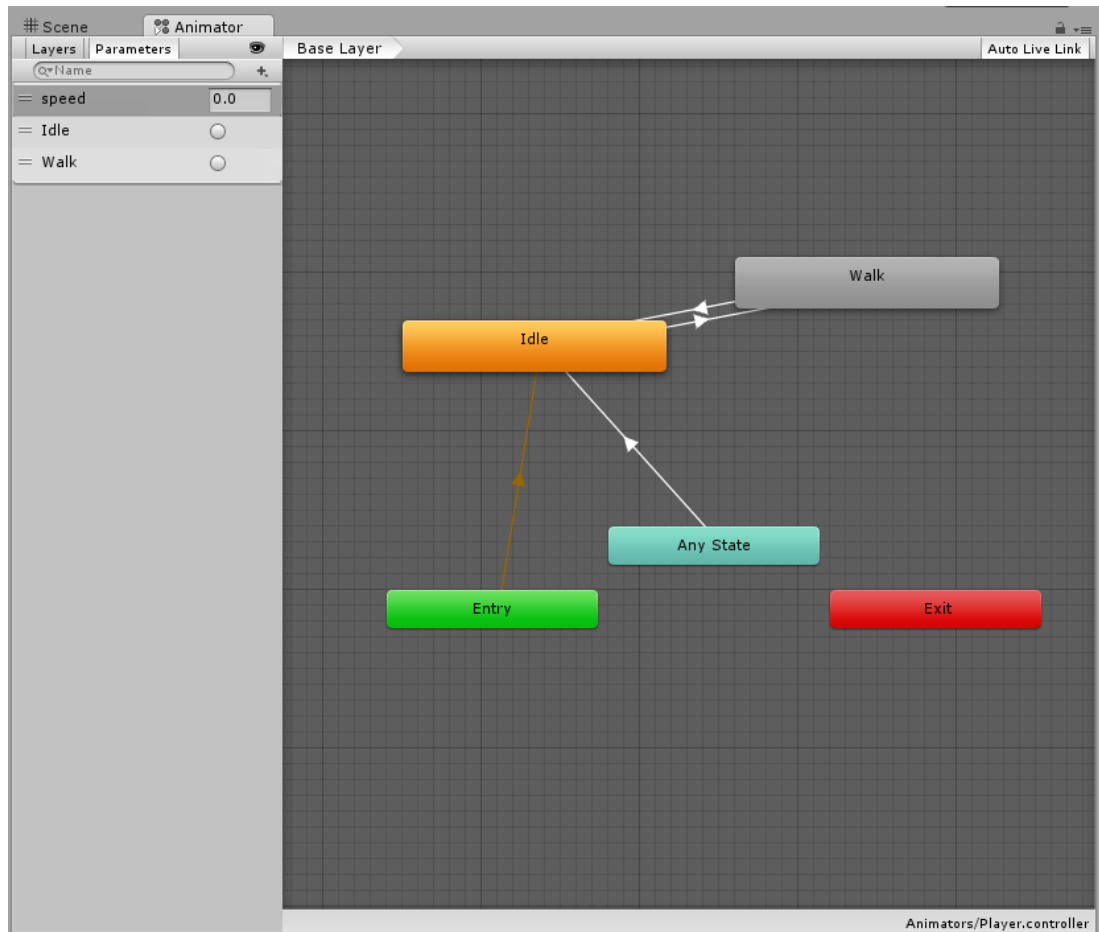
In the picture above we can see the multiple windows that offer different functionalities each, most important ones here being:

**Scene** – The scene window shows the visual rendering of the objects inserted into the scene in the Hierarchy window.

**Hierarchy** – The Hierarchy window shows the objects that exist in the scene. Here you can remove and add new objects and basically shape the game/project in whatever way Unity allows for through the Inspector.

**Inspector** – The Inspector window shows all the modifiable variable and scripts assigned to a selected game object from the **Hierarchy** window. Here you can assign scripts to them and modify their transform, material and other variables depending on the type of object.

### 3.2 Unity Animation.

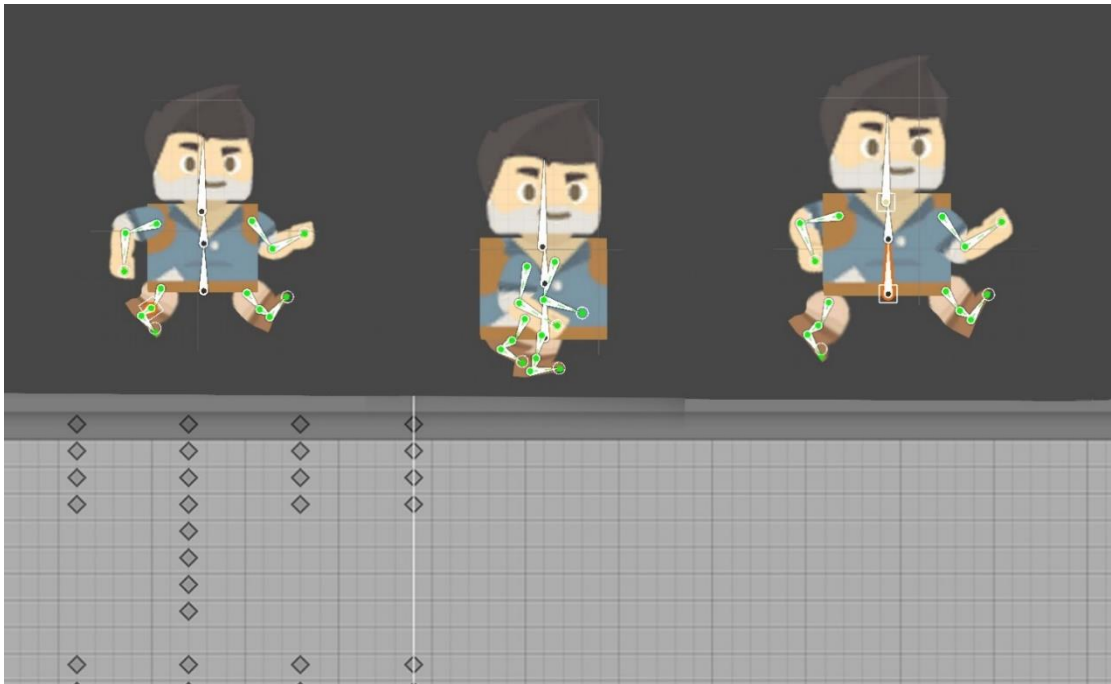


*Figure 9. Unity Animator screenshot, illustrating animation states and their conditions.*

**Animation States** are the basic building blocks of an **Animation State Machine**.

Each state contains an individual animation sequence (or **blend tree**) which will play while the character is in that state. When an event in the game triggers a state transition, a character will be left in a new state whose animation sequence will then take over.



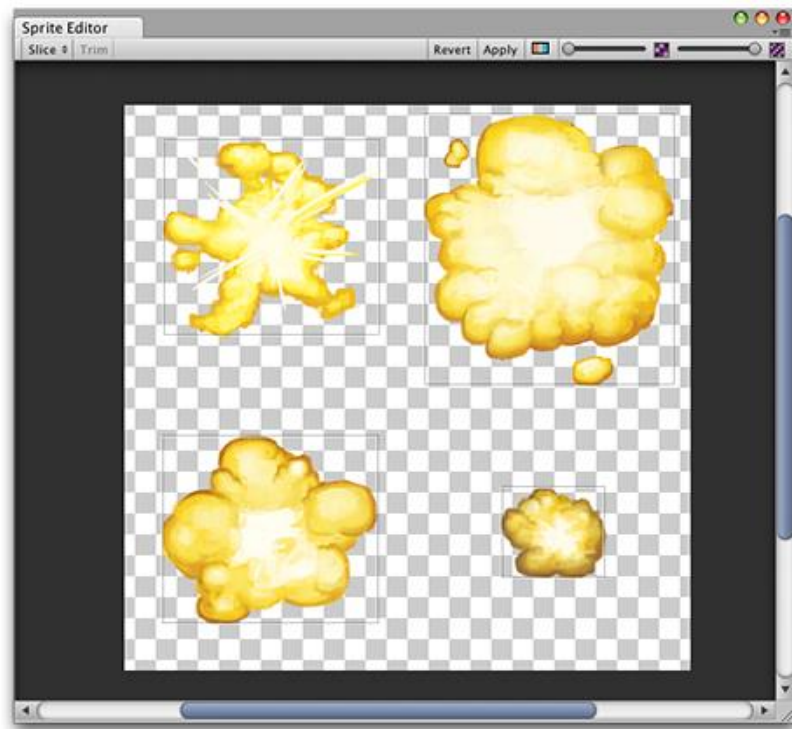


*Figure 10. Unity Animation screenshot, illustrating the animated object, it's bones/children game objects and the animation keyframes.*

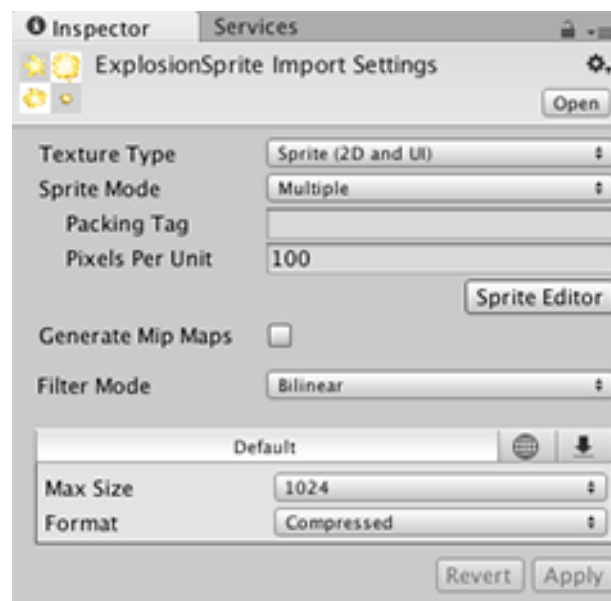
Unity's Animation features include retargetable animations, full control of animation weights at runtime, event calling from within the animation playback, sophisticated **state machine** hierarchies and transitions, blend shapes for facial animations, and much more.

### 3.3 Unity Sprite Editor

Sometimes a Sprite Texture contains just a single graphic element but it is often more convenient to combine several related graphics together into a single image. For example, an image could contain component parts of a single character, as with a car whose wheels move independently of the body. Unity makes it easy to extract elements from a composite image by providing a **Sprite Editor** for the purpose.



*Figure 11. Unity Sprite Editor screenshot, illustrating splitting a single sprite into different elements.*



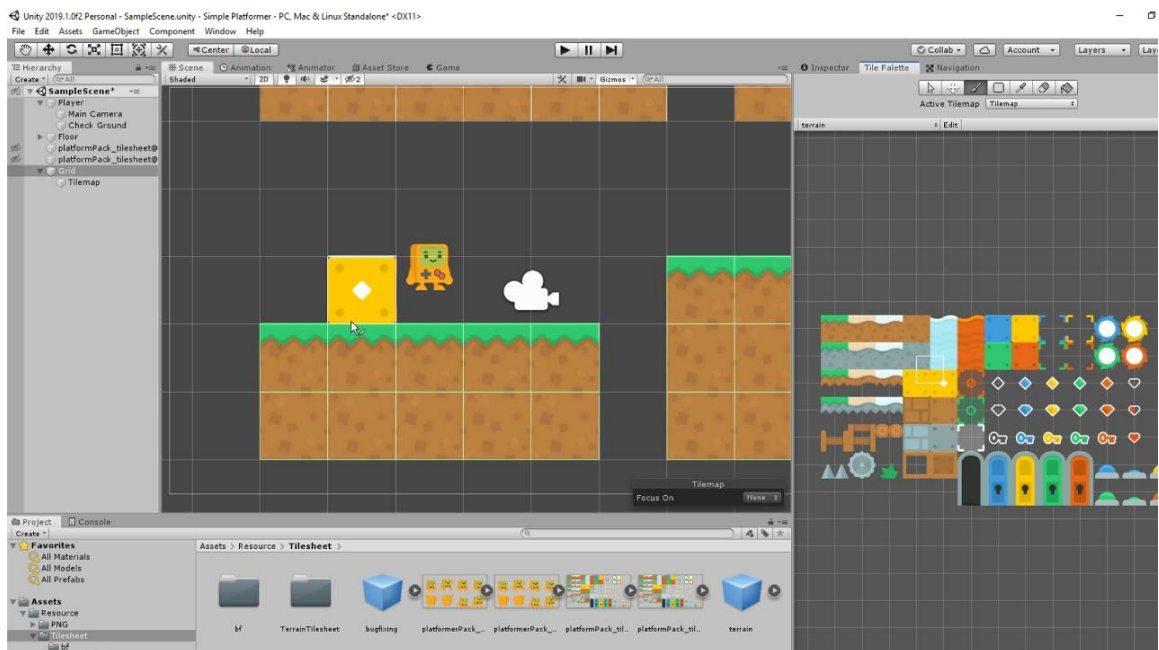
*Figure 12. Unity Sprite Editor screenshot, illustrating the import settings for the Sprite Editor.*

Through the **Inspector** window, we can see and edit the settings of the sprite editor import of a specific sprite that we select from the assets available in the project assets folder. In Sprite mode, we can specify whether a sprite image is actually a sprite sheet (multiple sprites in one image) or a single one.

In case of a sprite sheet (multiple image) clicking on the “Sprite Editor” button takes you to the **Sprite Editor** window where you can split images automatically or manually.

### 3.4 Unity Tilemap Palette

Unity Tile Palette is a 2D Tilemap System that allows to design 2D levels and maps based on a grid system while saving a huge amount of time. This system is free and built directly into the Unity editor and provides a plethora of features.



*Figure 13. Unity Tile Palette screenshot, illustrating the Tile Palette main window and functions.*

## 4 TECHNICAL GAME ANALYSIS

### 4.1 Overview

There are multiple systems at play here, and most of these are basic systems of this game genre. At this point, these are the major systems that exist in the present state of the game development.

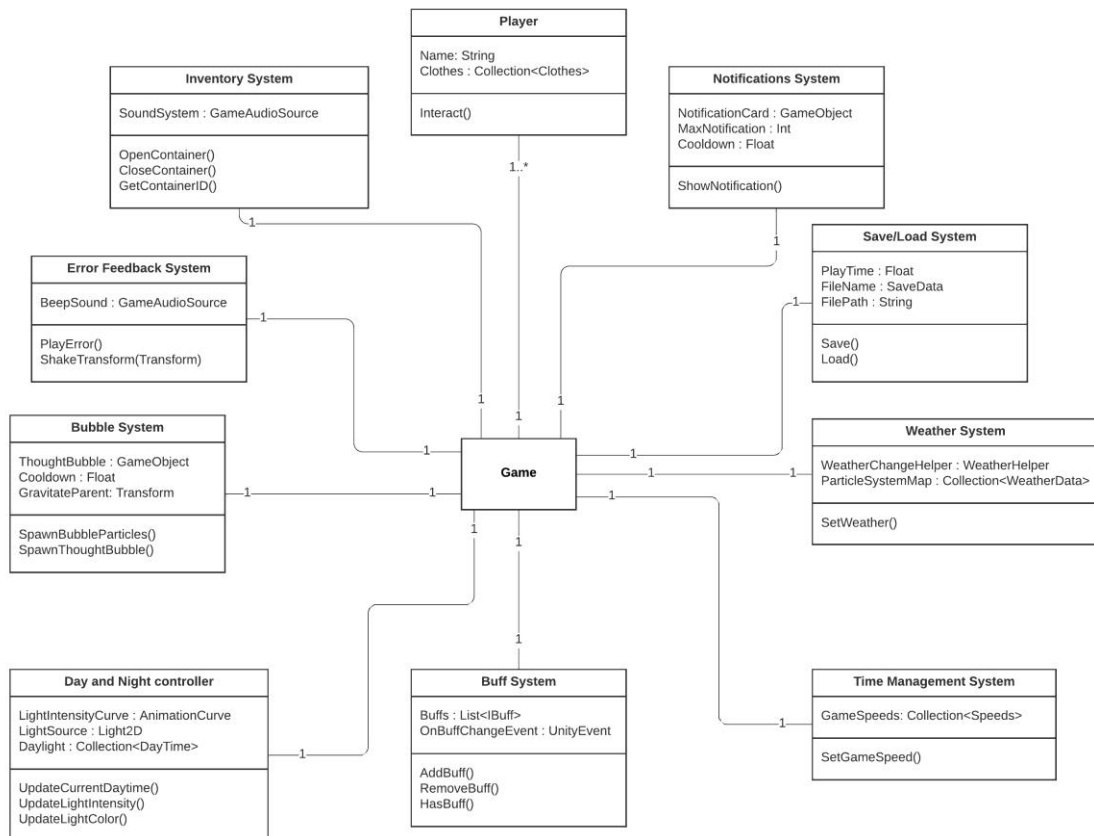


Figure 14. Scene Class diagram, illustrating the main game scene systems.

Let's explain each system individually

### 4.2 Systems

**Error Feedback System:** This system is responsible for notifying a player that some action went wrong such as not enough money to buy an item from a shop. In this case the **Error Feedback System** shakes the money canvas by simply calling its singleton's **ShakeMoneyError()** method.

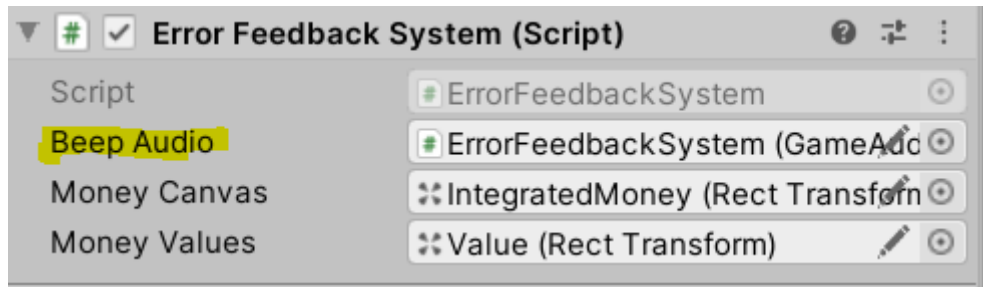


Figure 15. Unity Editor screenshot, showing the feedback system script and variables.

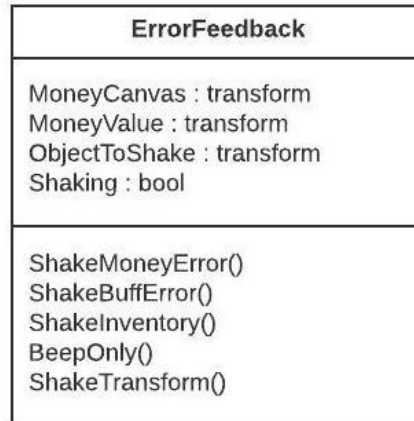


Figure 16. Error feedback class diagram, showing the error feedback system.

**Inventory System:** Every container in the game, including the player's inventory and containers such as fridges and wardrobes have their information displayed and managed by the inventory system.

In order to toggle a specific container to open or close simply call the method **OpenCloseContainer(ItemList list, string containerName)** from the Inventory Controller

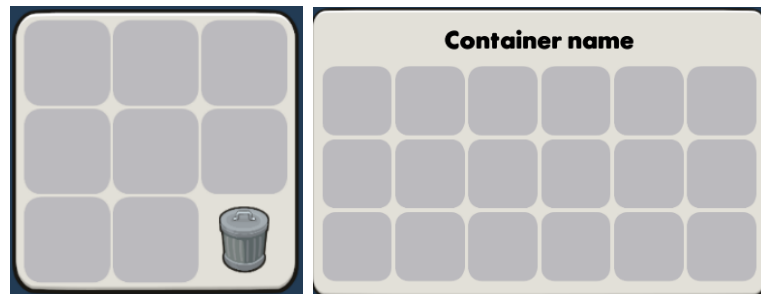


Figure 17. In game screenshot, showing the player inventory (on the left) and a random container (on the right).

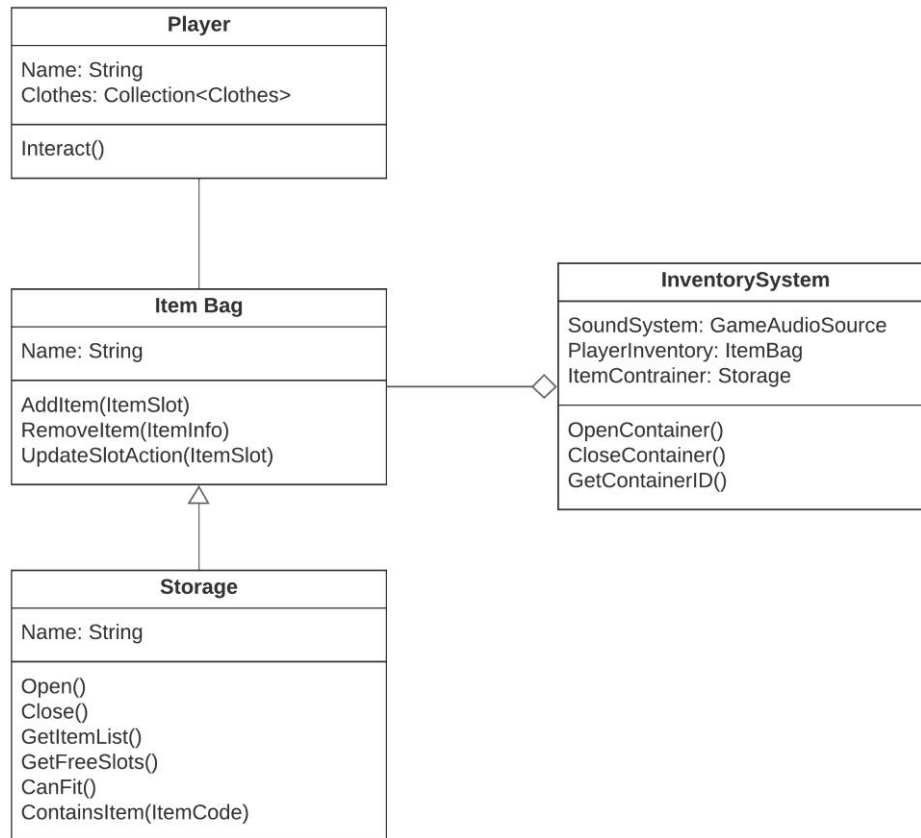


Figure 18. Player inventory Class diagram, illustrating inventory management.

**Notification System:** Shows a message notification on the screen by calling the method **CreateNotification**("Random Text", sprite).

It's also possible to show an icon with the message.

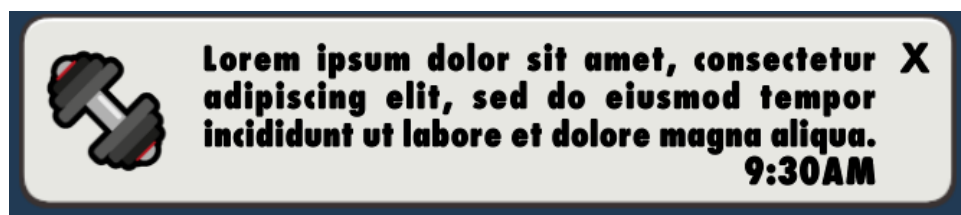


Figure 19. In game screenshot, showing a random test notification.

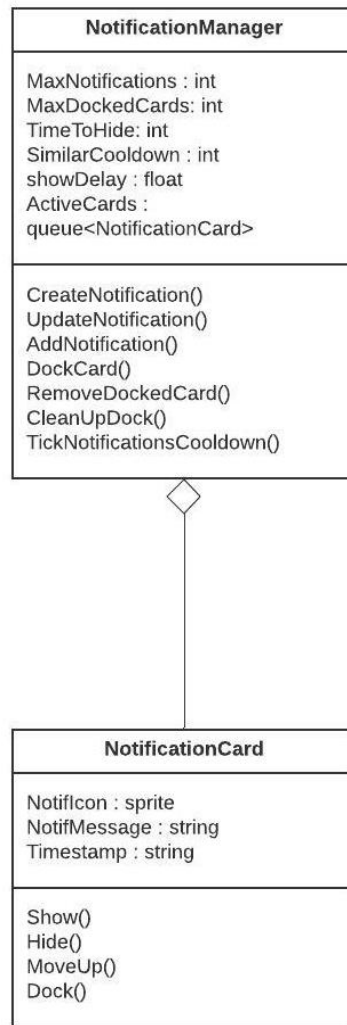


Figure 20. Notification class diagram, showing the notification system.

**Time Management System:** Responsible for setting and adjusting the game's speed depending on certain actions or through buttons on the respective UI.

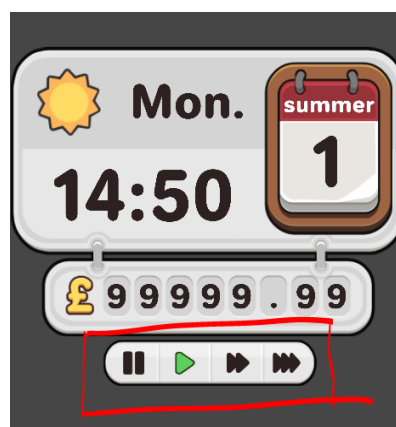


Figure 21. In game screenshot, showing the time management UI control buttons.

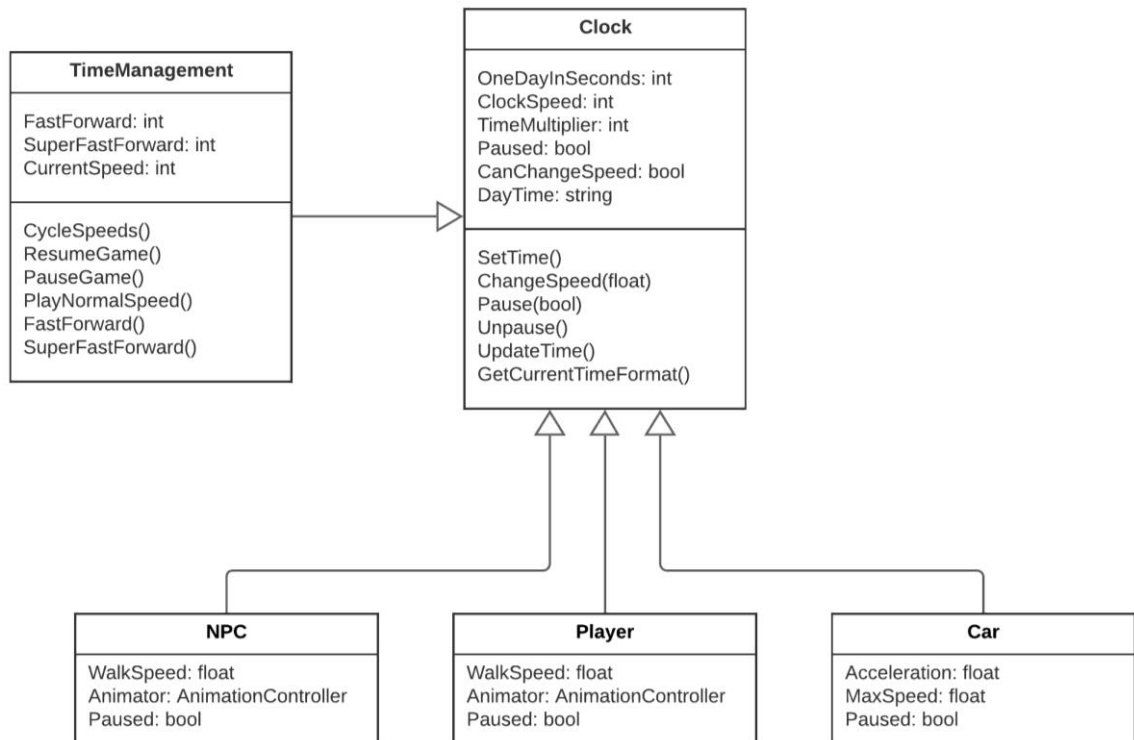


Figure 22. Time management class diagram, showing the time management system and game clock.

**Save and Load System:** This system is responsible for saving and loading the game data as well as player's saved game data including player skills, statuses, job history and job progress, world objects, furniture, building changes, etc..

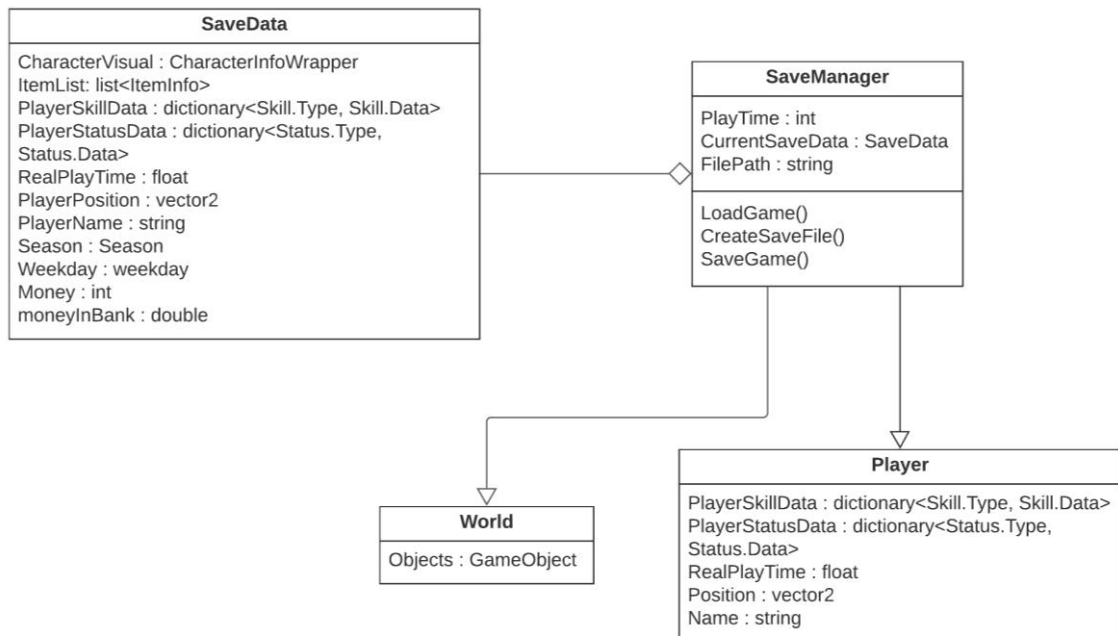
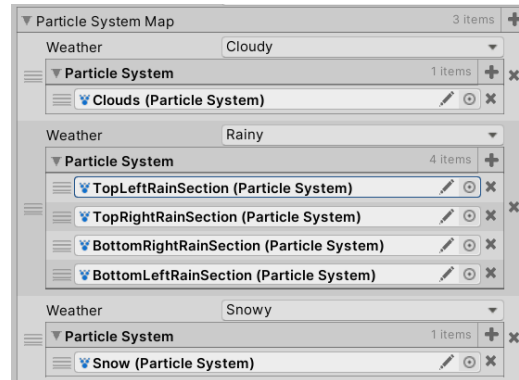


Figure 23. Save manager class diagram, showing the save/load system in the game.

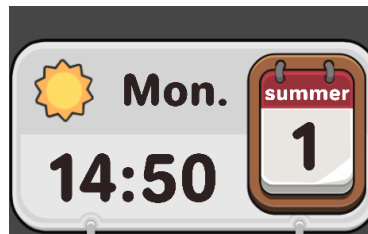


**Weather System:** Cycles between weathers created using scriptable objects after a random amount of in game time. It can cast weather particles if current weather has particles, pauses it or fade it off.



*Figure 24. Unity Editor screenshot, showing weathers and their particles assigned to the Particle System Map in the Weather System.*

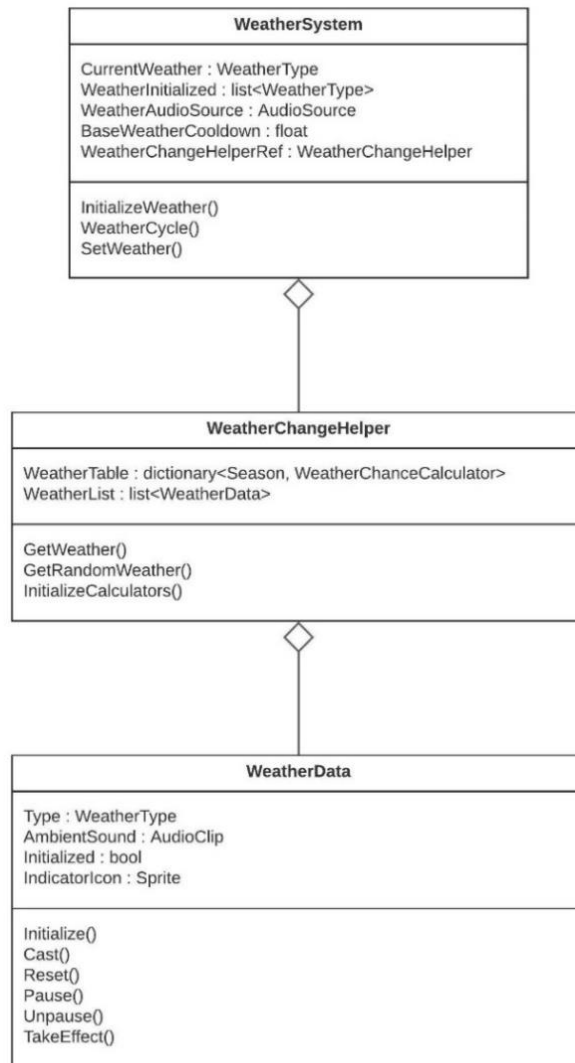
Weather icon is shown in the calendar date UI on the top left and a base particle effect is applied on weather start (sun rays, rain drops, clouds)



*Figure 25. Screenshot, showing the weather icon in the calendar UI.*

A set of pre-configured list of weathers with probabilities is created as a scriptable object and fed to the WeatherChangeHelper class to initialize and cycle through using the WeatherSystem.

Weathers can affect the player character's status or give bonuses or penalties by casting a buff or debuff on a player. This relation can be seen in the buff system section below.



*Figure 26. Weather system class diagram, showing the weather cycling system in the game.*

**WeatherData** is an abstract class that is used to make scriptable objects to create custom weather data with custom settings such as ambient sounds and effects. Currently, available weathers in the game are **Sunny**, **Rainy**, **Cloudy** and **Normal** and are cycled through using a preset probability table based on the seasons of the game. E.g. in Summer it's more likely to be sunny or normal compared to winter.

**Buff System:** Is responsible for adding or removing buffs from players, keeping tab on all current instances of buffs and debuffs given to the player.

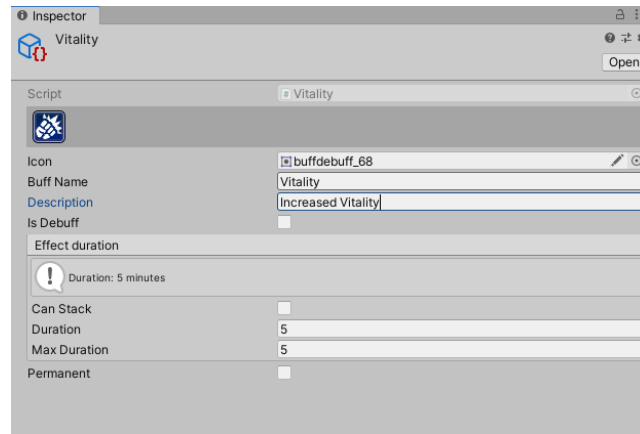


Figure 27. Unity Editor screenshot, showing a buff as a scriptable game object with different settings.

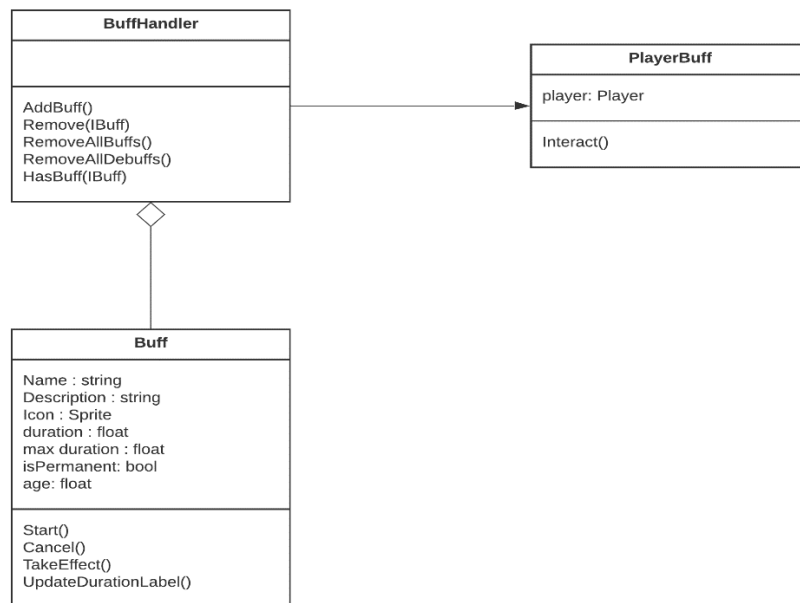
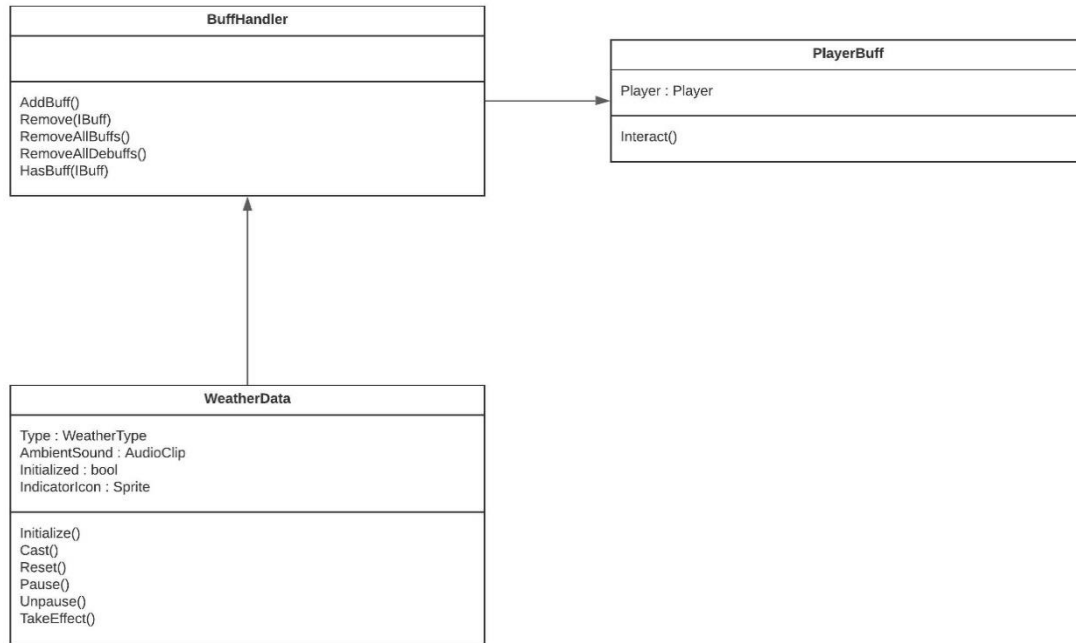


Figure 28. Buff class diagram, illustrating the character buff/debuff system.

The **Buff** class is an abstract class that is used to create scriptable objects to make buffs and debuffs with custom settings such as effect and duration. These buffs are then loaded from a resource folder inside Unity into BuffHandler and saved in the memory to be instantiated through specific events in the game.



*Figure*

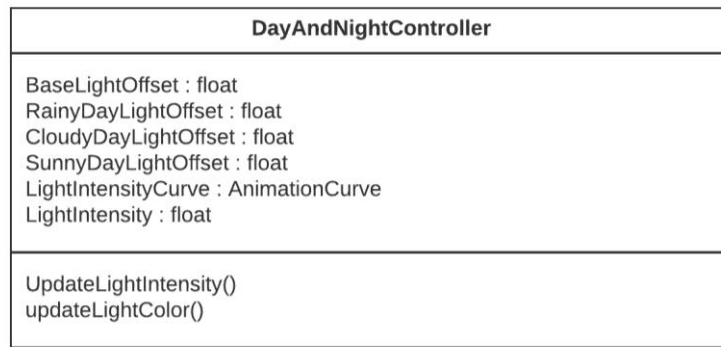
*29. Buff class diagram, illustrating the relation between the buff system and weathers.*

**WeatherData** can add or remove one or many buffs/debuffs on Cast and OnDestroy to the player.

**Day and Night:** Is responsible for the day and night cycle of the game and controls effects such as fog and scene lighting intensity.



*Figure 30. Ingame screenshots, showing day and night times of the day.*



*Figure 31. Day night class diagram, illustrating the day and night system controller.*

The **DayAndNightController** light intensity and color are affected by weather and season. In summer a day is longer and the sun rises earlier than in winter.

**Bubble System:** Spawns a bubble that plays an animation when one of player's statuses is very low as a reminder to do an activity that recovers it, otherwise if it falls below a specific threshold, a penalty is applied as a debuff.



*Figure 32. Ingame screenshot, Showing player statuses and warning status bubble.*

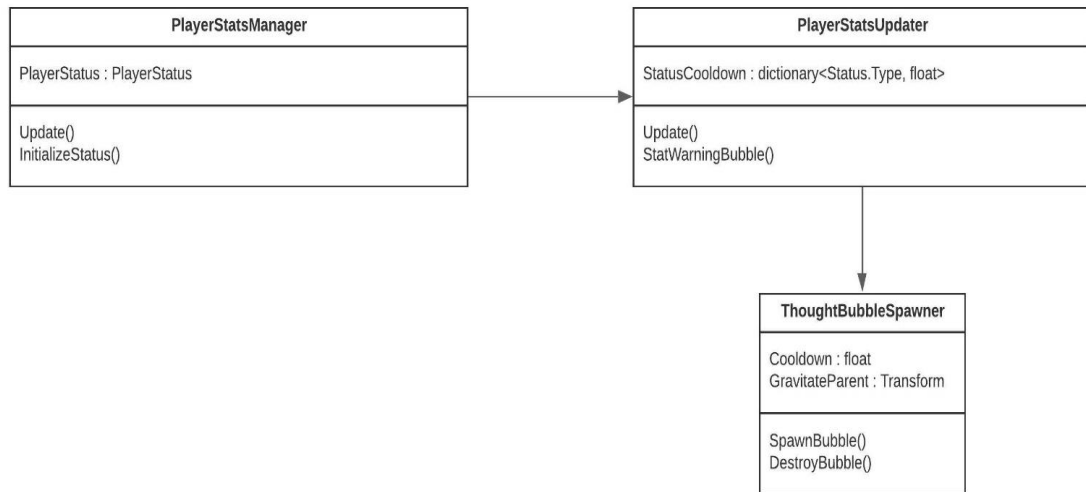


Figure 33. Bubble system class diagram, showing the bubble spawner system.

The **PlayerStatsManager** is the main script responsible for managing all player stats. Its **Update()** function is called every frame and in turn calls the **PlayerStatsUpdater** script's **Update()** function to update the player's statuses. Each of player's stats is reduced or increased by a specific amount or percentage over time and is affected by variables such as the weather and the activity being done (Running, sleeping, cooking).

### 4.3 Mechanics

**Calendar:** Stores information about a day, month, weeks and seasons and responsible for passing days and updating the Calendar UI.

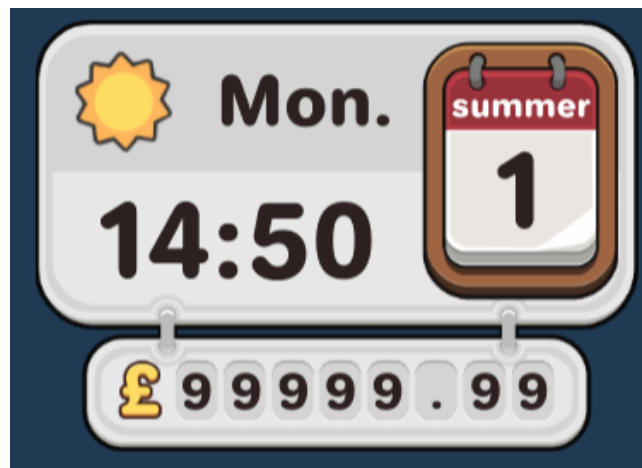


Figure 34. In game screenshot, showing the Calendar UI.

**Clock:** Is responsible for the game time related matters such as the game time relative to real time and pause/play functions.

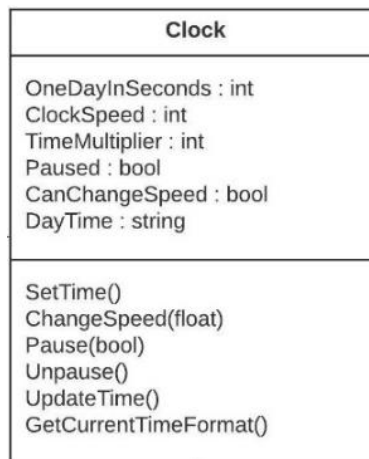


Figure 35. Clock class diagram, showing the clock class.

The in game time is faster than real life time, where 1 day in game is about 24 mins in real time. This is defined in the **Clock** class as the **OneDayInSeconds** variable. Also, a season is 30 days in game or 12 hours in real time.

**Player Stats and Skills:** Controlled by a manager, stats increase and drain according to different circumstances and actions and can be affected by external factors from other systems such as buffs/debuffs and weather.



Figure 36. In game screenshot, showing the player's Stat UI.



Figure 37. In game screenshot showing the player's Skills UI.

Some of the player skills are still in development. Those skills will be unlocked and used when we add a corresponding activity to said skill. At this stage, integrated skills are **Fitness, Logic, Cooking, Writing** and **Repairing** and are used in corresponding activities such as running on a treadmill, writing a novel, repairing some of the furniture in the house when they break, etc..

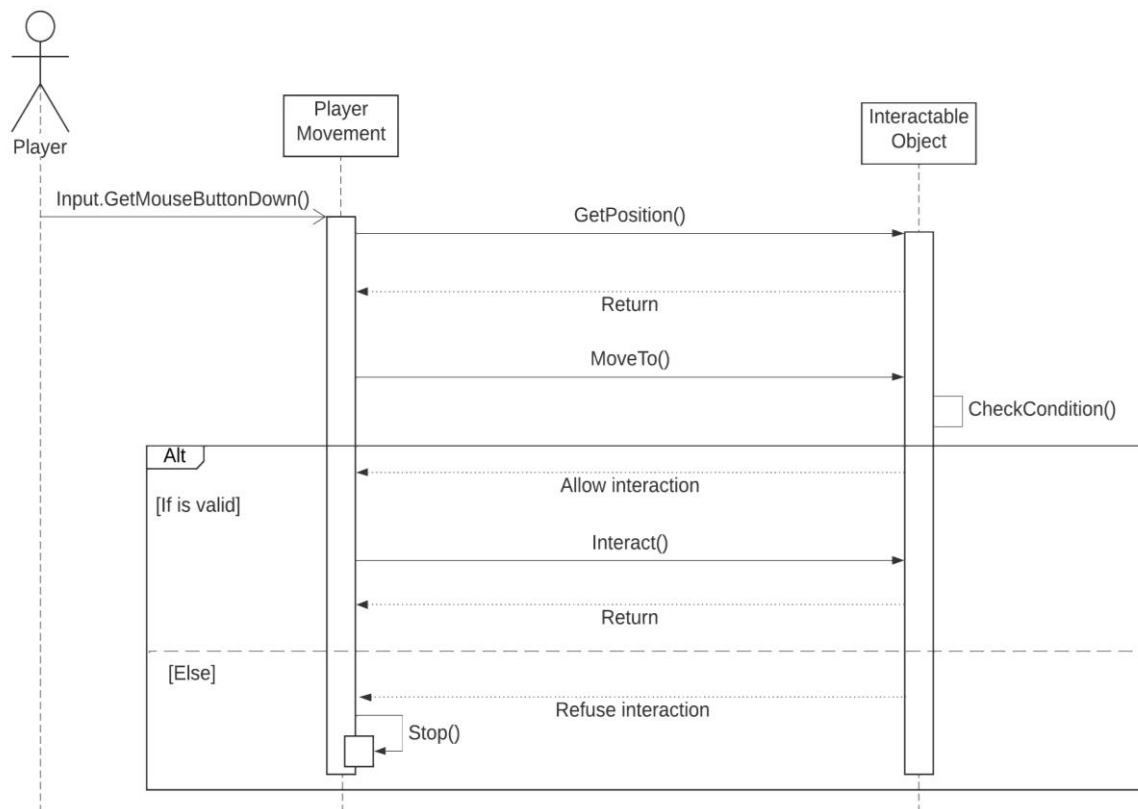
#### 4.4 Flows

Some of the interactions between systems in the game can be simple, while others are more complex and require multiple steps of functions to achieve.

Since Unity Engine is a graphic engine, it directly contains a graphic pipeline and a set of options that can be publicly called to change the visuals of the game.

These functions are called when the user makes changes through the settings menu displayed in either the main menu or the game scene.

Some of the options when set to a high level require a high-performance graphic card to process, while on low, should be able to run smoothly on the defined minimum setup in the non-functional requirements.

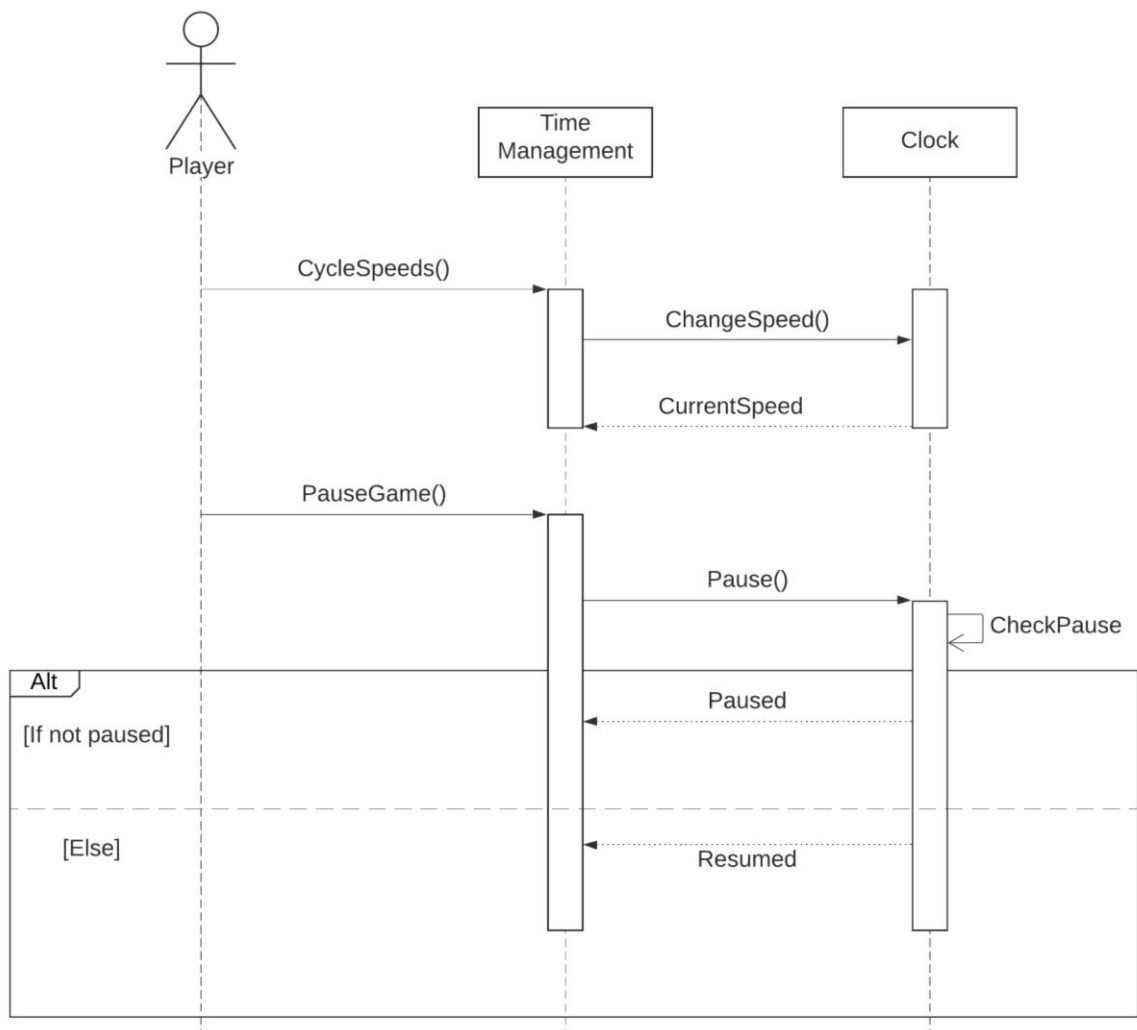




*Figure 38. Sequence diagram, showing the flow of interaction with interactable objects.*

All interactions with interactable objects in the game are lead with an intermediate player movement class that is responsible for movements, movements barriers, checks and interaction approvals.

Most interactable objects have general and specific conditions of use, for example the object should not be busy, broken or placed in a locked place or in a wrong position.

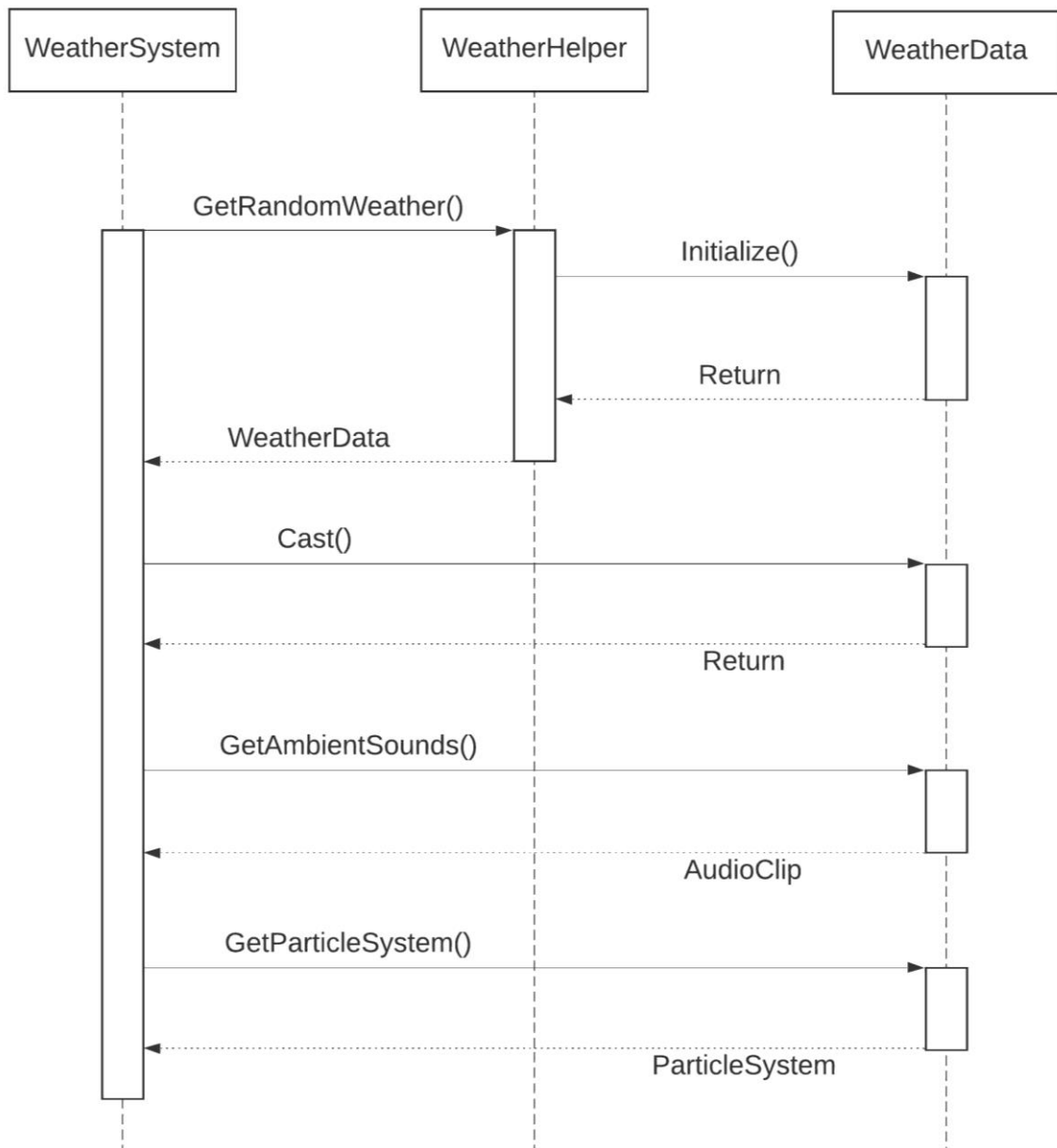


*Figure 39. Sequence diagram, showing the game speed cycling.*

Cycling between game speeds is called by user using the **Time Management** buttons or keyboard buttons.

The default speed is a variable set in the **Clock** class with a default value of 1, which is the normal speed relative to real time. Cycling through speeds sets the

game to faster speeds. The game can be paused or un-paused as well using its respective preset button which sets the game speeds to 1 (resumed) and 0 (paused).



*Figure 40. Sequence diagram, showing the weather cycle*

The **Weather System** periodically sets a new weather from the available weathers of the current season in the game.

The **Weather System** calls the GetRandomWeather() from the **Weather Helper** that contains a table of probabilities of weathers based on seasons, and returns a weather that inherits from **Weather Data**, then casts the weather special effects if it has any, followed by playing its specific ambient sounds and particles.

## CONCLUSION

As part of this work, the base functionalities and content of this genre has been developed in this game successfully. Most set tasks corresponding to the implementation of the formulated requirements were solved. The potential to expand the developed solution allows the possible further development of the functionalities and content of the sim game. The pre-alpha version of the game has been released on **Patreon** and the feedback was overwhelmingly positive.

## REFERENCES

1. Unity // Wikipedia. – [S. l.], 2021. – URL:  
[https://en.wikipedia.org/wiki/Unity\\_\(game\\_engine\)](https://en.wikipedia.org/wiki/Unity_(game_engine)) (access date: 01.05.2021)
2. Steam // Wikipedia. – [S. l.], 2021. URL:  
[https://en.wikipedia.org/wiki/Steam\\_\(service\)](https://en.wikipedia.org/wiki/Steam_(service)) (access date: 01.05.2021)
3. GameObject // Unity docs. – [S. l.], 2021. URL:  
<https://docs.unity3d.com/ScriptReference/GameObject.html> (access date: 01.05.2021)
4. Particle System // Unity docs. – [S. l.], 2021. URL:  
<https://docs.unity3d.com/ScriptReference/ParticleSystem.html> (access date: 01.05.2021)
5. Tilemap // Unity docs. – [S. l.], 2021. URL:  
<https://docs.unity3d.com/Manual/Tilemap-Isometric-SpritesImport.html> (access date: 01.05.2021)  
ScriptableObject // Unity docs. – [S. l.], 2021. URL:  
<https://docs.unity3d.com/Manual/class-ScriptableObject.html> (access date: 01.05.2021)
6. Unity Animation // Unity docs. – [S. l.], 2021. URL:  
<https://docs.unity3d.com/Manual/AnimationSection.html> (access date: 01.05.2021)
7. Unity Sprite Editor // Unity docs. – [S. l.], 2021. URL:  
<https://docs.unity3d.com/Manual/SpriteEditor.html> (access date: 01.05.2021)
8. Unity Tilemap Palette // Unity docs. – [S. l.], 2021. URL:  
<https://docs.unity3d.com/Manual/Tilemap-Palette.html> (access date: 01.05.2021)

## APPENDICE A ADDITIONAL ILLUSTRATIONS

### Launching the game (main menu)

In the main menu, we can change the game's video and audio settings, load a game, proceed to the character creation through the new game button or exit the game.



*Figure 41. Main menu screenshot*

Clicking on the settings menu opens the settings panel.

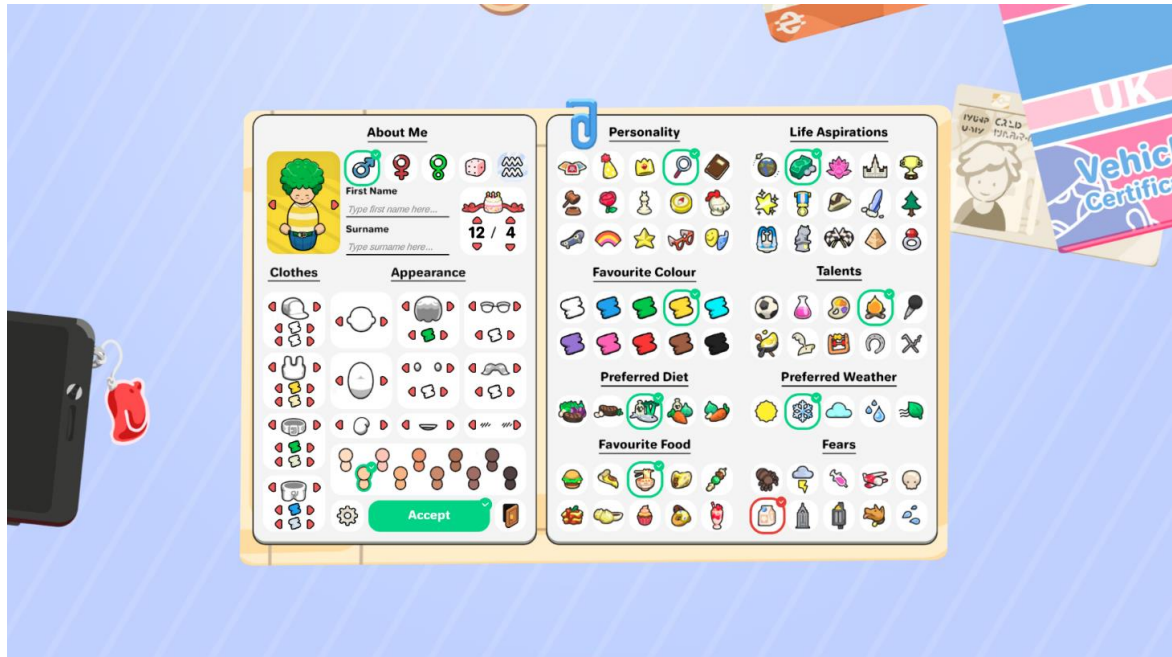


*Figure 42. Main menu screenshot, showing the settings panel.*

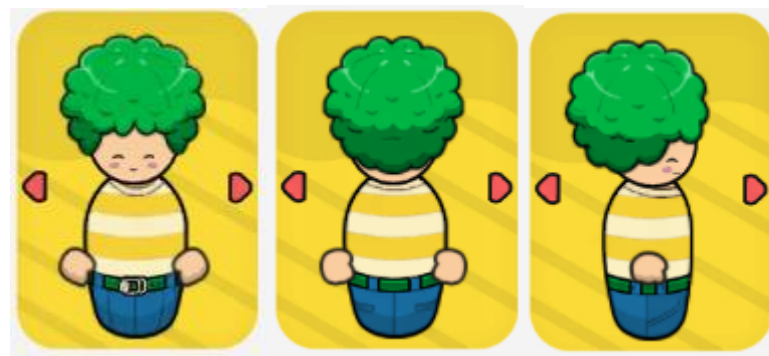
### Creating a character (main menu, character creation)

In the character creation, we can choose the character's gender, body parts,

clothes and clothes' color and name. We can also rotate the player.



*Figure 43. Character creation screenshot.*



*Figure 44. Character creation screenshot, showing different sides of the player character.*

## **Entering the game world scene**

Just after the main game scene, we go straight to the game world scene where we can start interacting with the game world and shape the character's life in whichever way we want.

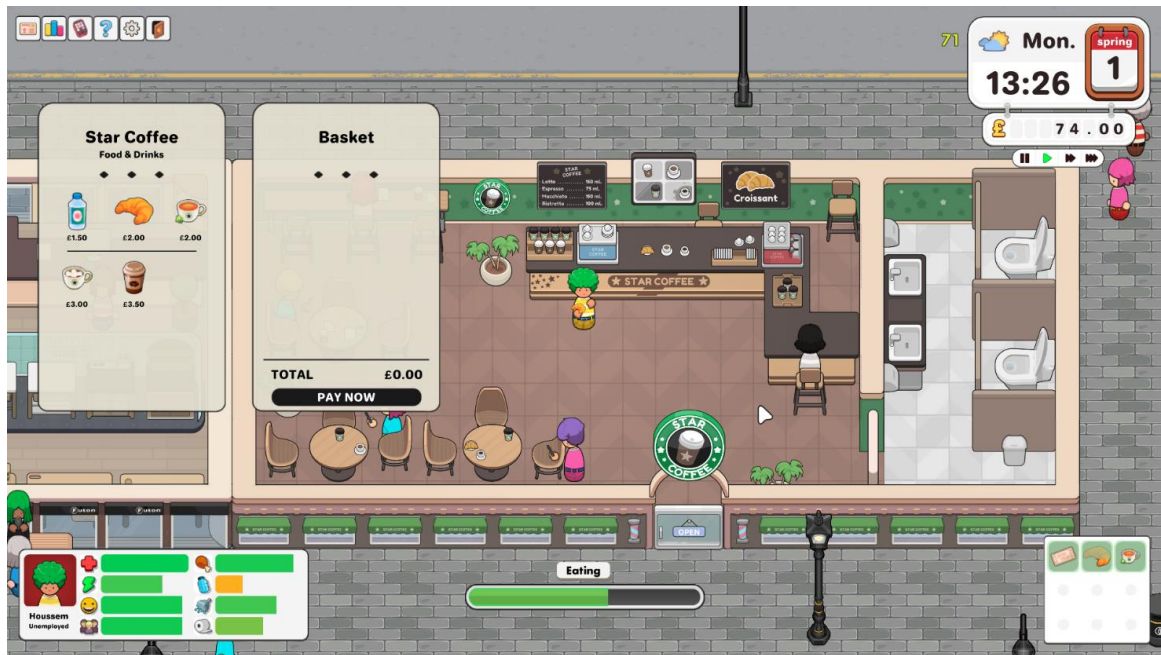




Figure 45. In game screenshot, showing the player's house in the game world.



Figure 46. In game screenshot, showing the player's house in the game world.



*Figure 47. In game screenshot, showing the player's house in the game world.*

Almost every object in the game world is interactable, including the furniture shown in this picture as well as NPCs (non-player characters) which are generated randomly and roam the game world, contributing in different ways to the player's gameplay experience.



# Отчет о проверке на заимствования №1



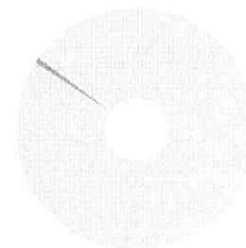
Автор: Neffati Housseem  
Проверяющий: (housseem.neffati@outlook.com / ID: 9197071)  
Отчет предоставлен сервисом «Антиплагиат» - [users.antiplagiat.ru](https://users.antiplagiat.ru)

## ИНФОРМАЦИЯ О ДОКУМЕНТЕ

№ документа: 2  
Начало загрузки: 04.06.2021 15:29:34  
Длительность загрузки: 00:00:01  
Имя исходного файла: thesis final.pdf  
Название документа: BACHELOR'S THESIS  
CREATING 2D GAME "LITTLE SIM WORLD"  
USING UNITY ENGINE  
Размер текста: 40 кБ  
Тип документа: Дипломная работа  
Символов в тексте: 41273  
Слов в тексте: 6649  
Число предложений: 354

## ИНФОРМАЦИЯ ОБ ОТЧЕТЕ

Начало проверки: 04.06.2021 15:29:36  
Длительность проверки: 00:00:18  
Корректировка от 04.06.2021 15:36:23  
Комментарии: не указано  
Модули поиска: Интернет



**ЗАИМСТВОВАНИЯ**  
0,95%

**САМОЦИТИРОВАНИЯ**  
0%

**ЦИТИРОВАНИЯ**  
0%

**ОРИГИНАЛЬНОСТЬ**  
99,05%

Заимствования — доля всех найденных текстовых пересечений, за исключением тех, которые система отнесла к цитированиям, по отношению к общему объему документа.  
Самоцитирования — доля фрагментов текста проверяемого документа, совпадающий или почти совпадающий с фрагментом текста источника, автором или соавтором которого является автор проверяемого документа, по отношению к общему объему документа.  
Цитирования — доля текстовых пересечений, которые не являются авторскими, но система посчитала их использование корректным, по отношению к общему объему документа. Сюда относятся оформленные по ГОСТу цитаты; общеупотребительные выражения; фрагменты текста, найденные в источниках из коллекций нормативно-правовой документации.  
Текстовое пересечение — фрагмент текста проверяемого документа, совпадающий или почти совпадающий с фрагментом текста источника.  
Источник — документ, проиндексированный в системе и содержащийся в модуле поиска, по которому проводится проверка.  
Оригинальность — доля фрагментов текста проверяемого документа, не обнаруженных ни в одном источнике, по которым шла проверка, по отношению к общему объему документа.  
Заимствования, самоцитирования, цитирования и оригинальность являются отдельными показателями и в сумме дают 100%, что соответствует всему тексту проверяемого документа.  
Обращаем Ваше внимание, что система находит текстовые пересечения проверяемого документа с проиндексированными в системе текстовыми источниками. При этом система является вспомогательным инструментом, определение корректности и правомерности заимствований или цитирований, а также авторства текстовых фрагментов проверяемого документа остается в компетенции проверяющего.

№	Доля в отчете	Источник	Актуален на	Модуль поиска
[01]	0,69%	Life simulation game <a href="http://en.wikipedia.org">http://en.wikipedia.org</a>	22 Авг 2020	Интернет
[02]	0,26%	Электронный практикум по созданию игр на платформе Unity3d <a href="https://core.ac.uk">https://core.ac.uk</a>	31 Окт 2020	Интернет