TARTU UNIVERSITY

FACULTY OF SOCIAL SCIENCES

NARVA COLLEGE

INFORMATION TECHNOLOGY SYSTEMS DEVELOPMENT

Anton Laagus

A STUDY OF THE USE OF 3D INTERACTIVE GAME SOLUTIONS FOR LOGIC DEVELOPMENT FOR SCHOOL STUDENTS.

Diploma thesis

Supervisor: Dr. Chen-Wan Lin

NARVA 2023

Olen koostanud töö iseseisvalt. Kõik töö koostamisel kasutatud teiste autorite tööd, põhimõttelised seisukohad, kirjandusallikatest ja mujalt pärinevad andmed on viidatud. .....
Töö autori allkiri ja kuupäev

Contents

TERMS AND ABBREVIATIONS INTRODUCTION

VFX – Visual effects.

IK - Inverse Kinematics

C# – object-oriented programming language.

Script – C# class in Unity.

Collider - the shape of an object for the purposes of physical collisions.

Shader - small script that calculates color of each pixel, based on the lighting input.

Rigging – 3D model bones arrangement in order to make the model move.

Weight painting – assigning certain areas of the model to certain bones.

INTRODUCTION

The problem

Nowadays, children and school students devote less and less time to solving various puzzles. They prefer activities that are more amusing to do. Because of this, even the simplest puzzle may seem very difficult or boring to solve for them. They get tired or bored quickly, because they cannot solve them, or puzzles might seem dull and not entertaining.

The solution

One of the solutions to this problem might be an interactive game where a player would be given an opportunity to solve various puzzles in order to complete the game. Puzzles could be of varying degrees of difficulty, so the player has to concentrate on the game process if they want to advance to the next stage of the game. However, those types of game should not be very long, because at some point the player will get bored and distracted due to long concentration.

The aim

The aim of the work is to develop a 3D game. The player will have to control a character and interact with in-game objects in order to solve various puzzles, such as logic puzzles or visual puzzles, where the player has to repeat a sequence of the indicated objects. Those types of puzzles can improve one's memory or attentiveness. The goal of the game is to get to the very end. The player will not be able to reach the end if they decide to skip some of the major puzzles.

The motivation

The author has an interest in the field of game development. The author will go through the whole process of game development with no experience in this field. While doing so, the author will try his hand in different aspects of game development, such as Visual effects, shaders, 3D modeling, animation and much more.

Tasks

Task is to create the game where author has to implement the following tasks:

- Create character model, texture, rig, animations.
- Come up with various puzzles.
- Come up with game mechanics.
- Game level design.
- Create shaders for objects, such as dissolve effect for chest, water shader etc.
- Create animations for different non-character objects.
- Create VFX for characters and other different objects.
- Build a game level.
- Interaction with in-game objects, such as chests etc.
- Proper scene lighting.
- Post processing.
- Test application.

Outline

The first section contains an introduction with the main objectives of the project. In this section, the author describes the existing problem and proposes a solution.

In the second section, the author lists the technologies and tools used in the work. The author describes the system development process. In addition, the author describes the future development of the application.

Finally, the third section includes the references and the bibliography used in the project.

IMPLEMENTED TECHNOLOGIES AND AVAILABLE SOLUTIONS

Technologies

Unity

Unity is a cross-platform game engine developed by Unity Technologies, first announced and released in June 2005 at Apple Inc.'s Worldwide Developers Conference as a Mac OS X-exclusive game engine. As of 2018, the engine had been extended to support more than 25 platforms. The engine can be used to create three-dimensional, two-dimensional, virtual reality, and augmented reality games, as well as simulations and other experiences. The engine has been adopted by industries outside video gaming, such as film, automotive, architecture, engineering and construction. [1]

Blender

Blender is the free and open-source 3D creation suite. It supports the entirety of the 3D pipeline—modeling, rigging, animation, simulation, rendering, compositing and motion tracking, even video editing and game creation. [2]

C#

C# (pronounced "See Sharp") is a modern, object-oriented, and type-safe programming language. C# enables developers to build many types of secure and robust applications that run in the .NET ecosystem. C# has its roots in the C family of languages and will be immediately familiar to C, C++, Java, and JavaScript programmers. [3]

Paint.NET

Paint.net (stylized as Paint.NET or paint.net) is a freeware raster graphics editor program for Microsoft Windows, developed on the .NET Framework. Paint.net was originally created by Rick Brewster as a Washington State University student project, and has evolved from a simple replacement for the Microsoft Paint program into a program for editing mainly graphics, with support for plugins. [4]

HLSL

The High-Level Shader Language or High-Level Shading Language (HLSL) is a proprietary shading language developed by Microsoft for the Direct3D 9 API to augment the shader assembly language, and went on to become the required shading language for the unified shader model of Direct3D 10 and higher. [5]

Visual Effect Graph

The Visual Effect Graph is a node based visual effect editor. It allows to author next generation visual effects that Unity simulates directly on the GPU. The Visual Effect Graph is production-ready for the High Definition Render Pipeline and runs on all platforms supported by it. [6]

Design and architecture of application

Instead of using traditional programming approach, where everything is a class, the author will use an idea that everything in the scene is a GameObject and almost all functionality is implemented as MonoBehavior attached to that game object.

The MonoBehaviour class provides the framework, which allows attaching C# script (**Figure 1**) to the GameObject in the editor, as well as providing hooks into useful events.

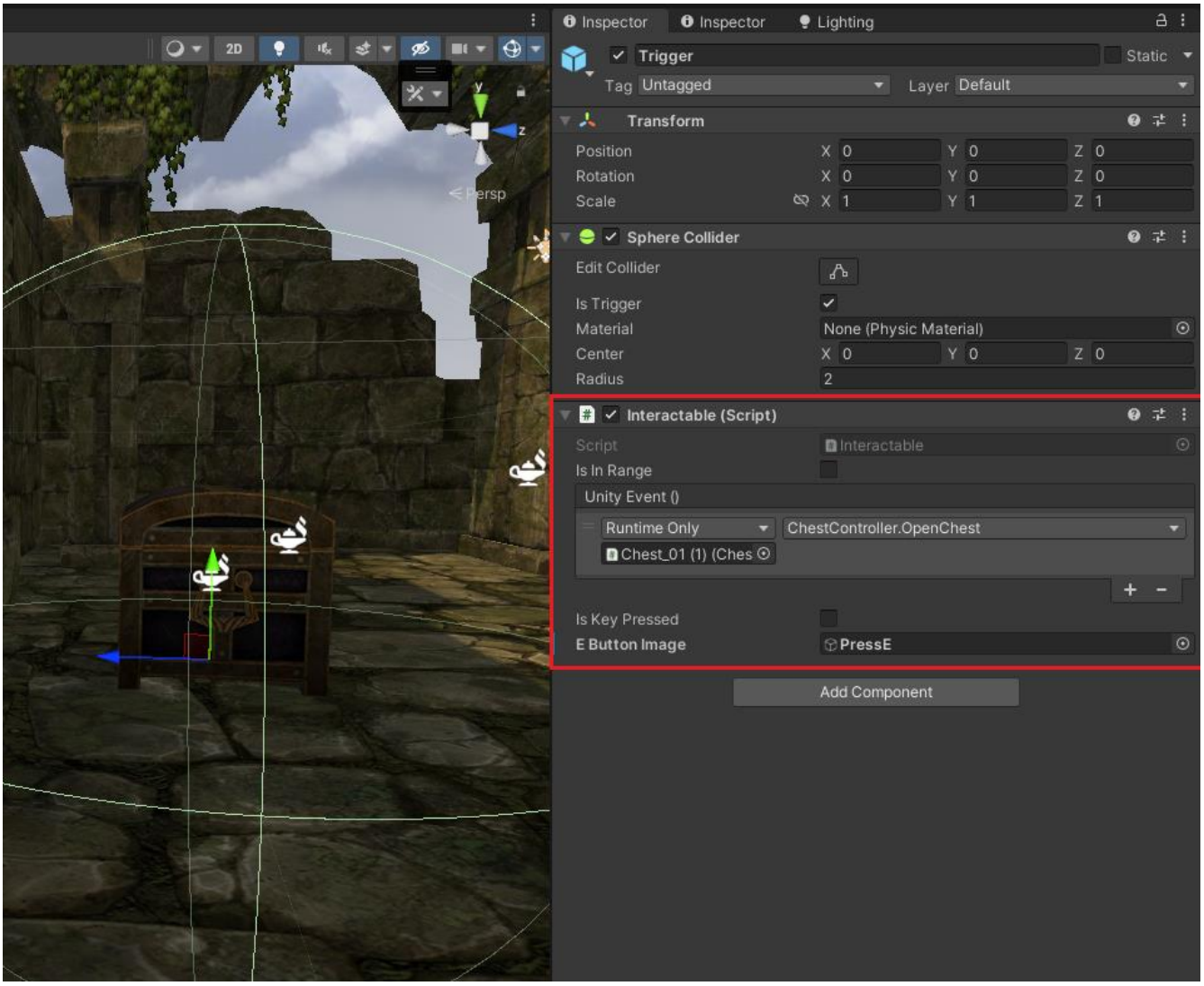

Figure 1. *GameObject with attached C# script.*

All files are stored in appropriate folders according to the naming convention. Character animations and other object animations are stored in Animations folder, materials stored in Materials folders, scripts stored in Scripts folder and so on.

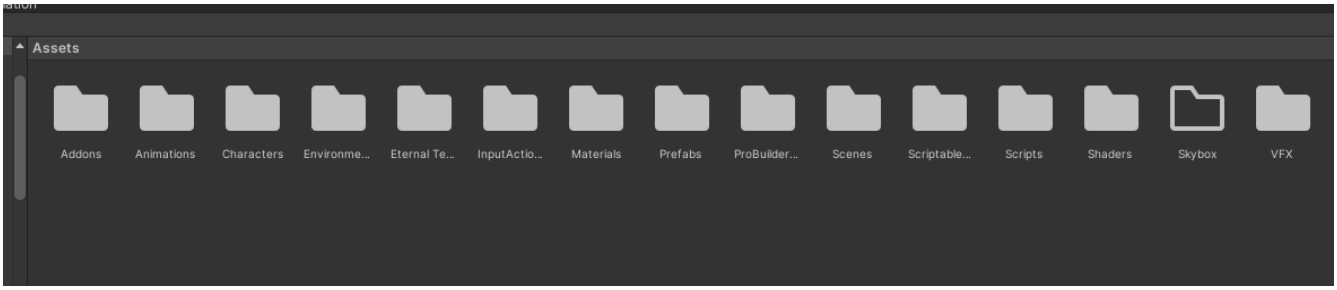

Figure 2. *Folder structure.*

Despite the fact that Unity allows to make games for different platforms, e.g., iOS, PS5, Android and others, the author decided to make the game just for Windows. Otherwise, it is necessary to make changes to the game input system and other changes.

Functional requirements

- The game must be developed in Unity.
- The game must be implemented with C# script.
- The game must be controlled with mouse and keyboard.
- The game must be played on PC.
- The game must have one level.
- The game must have different puzzles to solve, such as:
    - Find a bridge puzzle, where the player must find an invisible bridge by rotating a camera.
    - Maze-like puzzle, where the player must go through the maze and find an exit.
    - Color matching puzzle, where the player must find matching colored slime.
    - Repeat pattern puzzle, where the player must repeat pattern from the wall using buttons on the ground.
- The character must be able to walk, run and jump.
- The player must be able to interact with several in-game objects.

Non-functional requirement

- The game must run on Windows 10/11 and maintain the same behavior.
- The application must work stably.
- The game must not crash (it has been tested for 24 hours).
- The game must be user friendly; in most cases the player doesn't need other people's support.

Early prototype of application

The technologies used for the development of the early application prototype were Unity, C#, Blender, HLSL, Shader Graph, Visual Effect Graph and others. Currently, all the major features of the application are implemented: character, its texturing, weight painting and animating are done; game level is finished, the most puzzles to solve are completed.

The result of this thesis project is a finished application prototype, which has main features implemented. The functionality available right now in the application prototype gives a good overview of the features of the finished application.

The game prototype contains two scenes. The first scene is intended to visualize most of the progress. It contains all the main elements of the game.

The first scene (**Figure 3**) is a game level itself; all major game mechanics are presented here. It contains most of the features that were developed, but some ideas and puzzles did not make it through and therefore they were left in the second scene (**Figure 9**) for further revision.

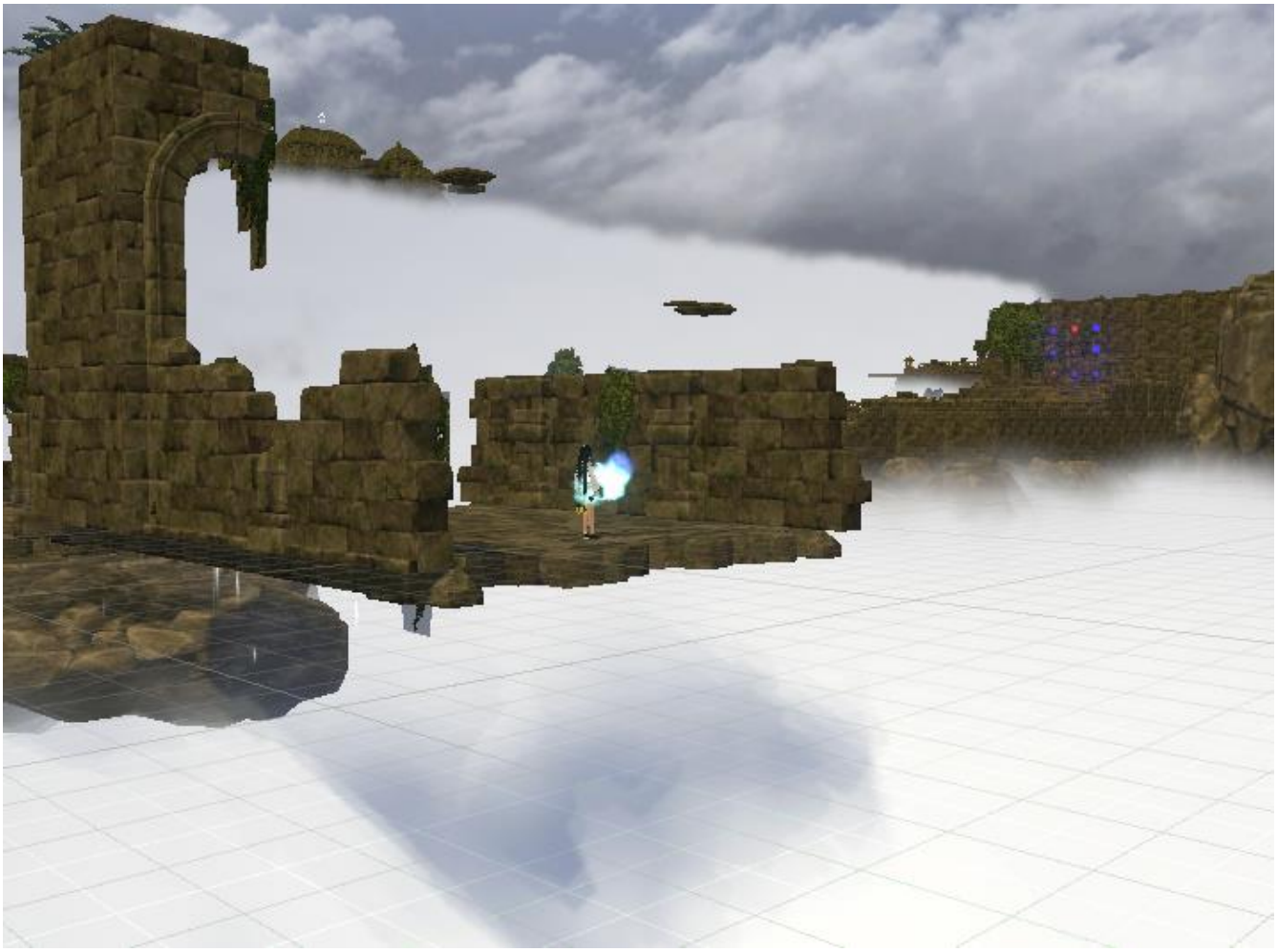

Figure 3. *The first scene.*

One of the first key puzzles the player encounters is a bridge search. The bridge can be found only at certain camera angles (**Figure 4**) and therefore the player has to move the character and rotate the camera around in order to find the bridge. The player cannot find it if they are standing still in one place and if the player does not notice the bridge, they will not be able to continue and progress to the further stages of the game. After the bridge is found, the player must use it to get to the other side. The difficulty lies

in the fact that the bridge is maze-like, and the player has to find the way out.

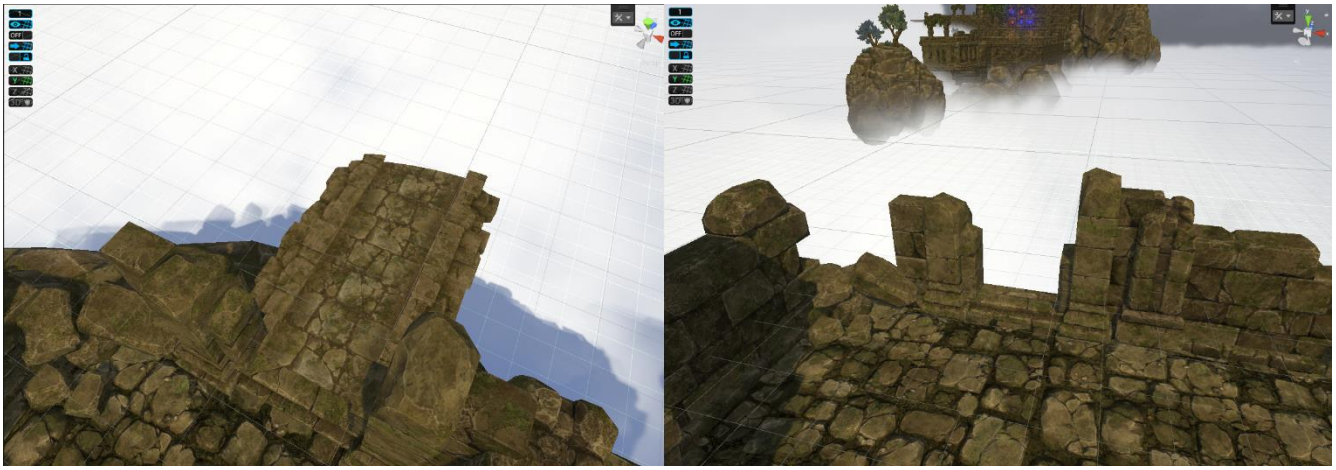

Figure 4. *Bridge puzzle.*

This (**Figure 5**) is another no less important puzzle. The player has to repeat the pattern on the wall. The player has to interact with the buttons that are located on the ground right beneath the wall and in order to activate them, they need to step on one of the buttons, and it will change its color from red to blue. If the player repeats the pattern from the wall correctly, then the door will automatically open, and the player will be able to get to the next stage.

Figure 5. *Repeat pattern puzzle.*

This puzzle (**Figure 6**) requires the player to find a slime of the same color that is currently located somewhere nearby. There are 2 buttons on the ground, the player needs to select the correct one. The correct button is the one that has a slime of the same color as one of the slimes next to it. Incorrect is the button on which a slime without a color pair stands. If the player has selected the correct button, then the next round starts. In each subsequent round, the number of slimes is added, which adds to the difficulty in finding the right color. If the player selects the wrong button, the round restarts automatically. Eventually, if the player selects the right button a certain number of times, a bridge will rise from the water, allowing the player to cross over to the other side. There is a possibility that the player may be colorblind, so this is the only puzzle that can be skipped by finding a passage near a large rock.

Figure 6. *Color matching puzzle.*

The last trial is the puzzle where the player has to move blocks that are hidden somewhere in the building to specific points in order to open the last door and get in-game reward. There are three points that need to be activated and three blocks the player can interact with. When the block the player brought is on the point then point visual effect changes its color from initial red to blue. If the player lost the block, e.g., block fell from a cliff, they have opportunity to reset this puzzle by pulling the lever and start over.

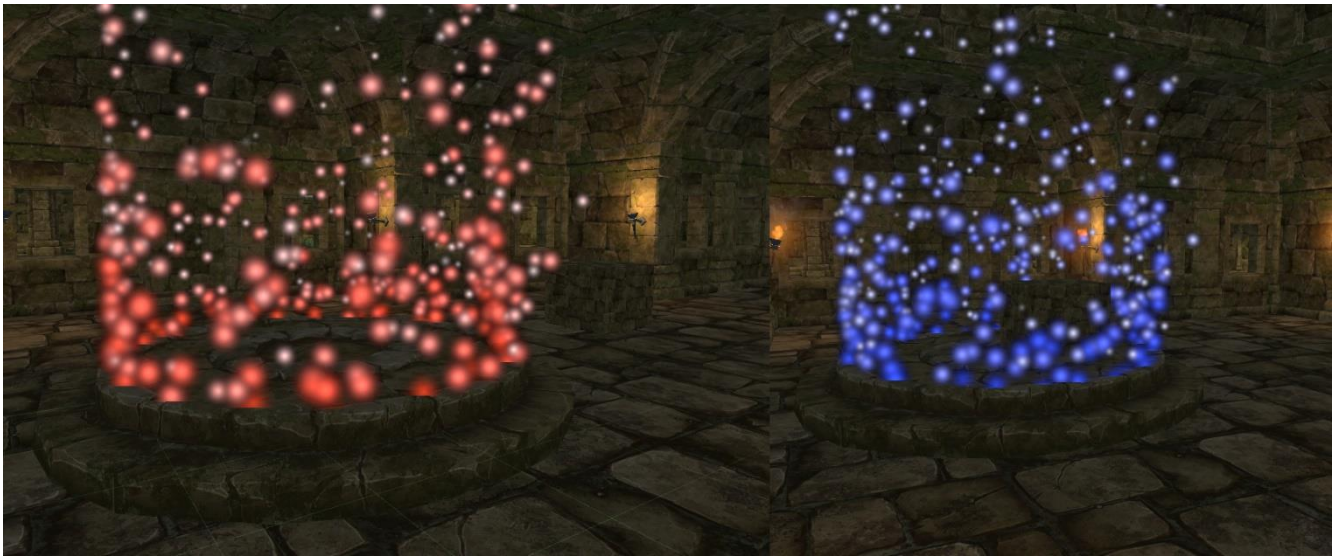

Figure 7. *Place the block puzzle.*

There are other smaller and less important puzzles that the player has to solve. Puzzles such as jumping off the platform, so an airflow will send the character to another side of the cliff or find a lever to raise a bridge from the water to the surface or stand on the platform, so it takes the character to the temple high in the sky.

One of the other important core mechanics in the game is a resurrection of the player if they fall down at one of the stages of the game. To make it possible, the game has a feature to save one's progress by activating saving points (**Figure 8**). The save point is considered activated if the player has reached the key zone of the game and has come close enough to the saving point. When the player gets to the saving point for the first time, a hint will appear with an explanation of how it works providing image and text. Additionally, when the player is close enough to the saving point then the visual effect will appear.

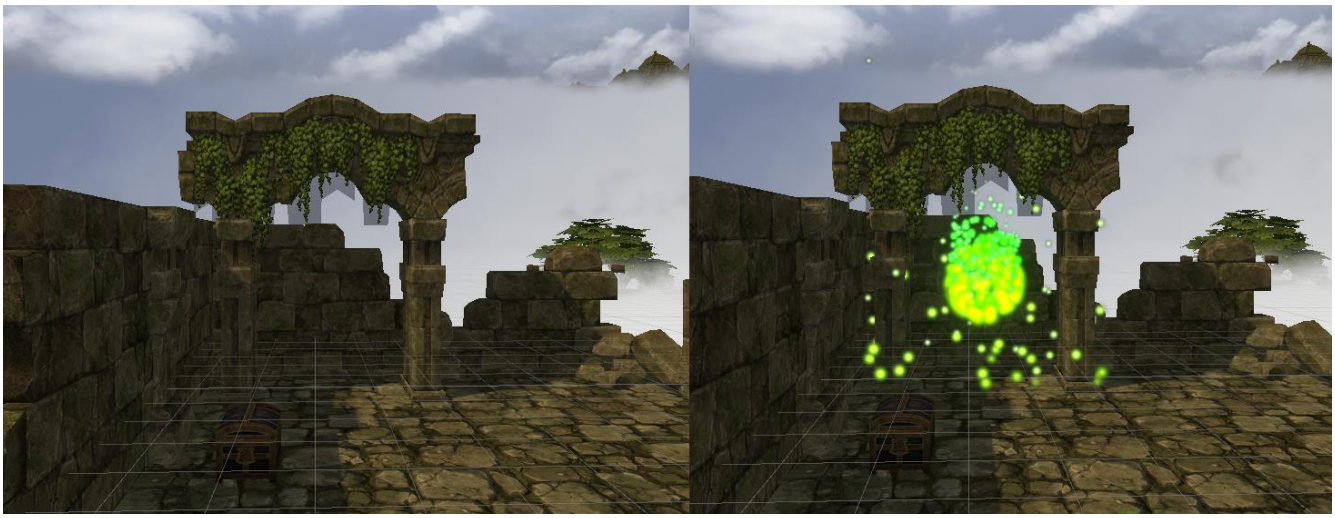

Figure 8. *Resurrection point.*

The second scene serves as a testing ground. In this scene (**Figure 9**), a check is made whether the shaders, visual effects, scripts, materials, animations and other aspects of the game are suitable for adding them to the main scene. If they do not meet criteria for adding them and making them part of the game,

they stay in the second scene and undergo some changes if it is possible. The player should not see this scene because its sole purpose is for testing new features.

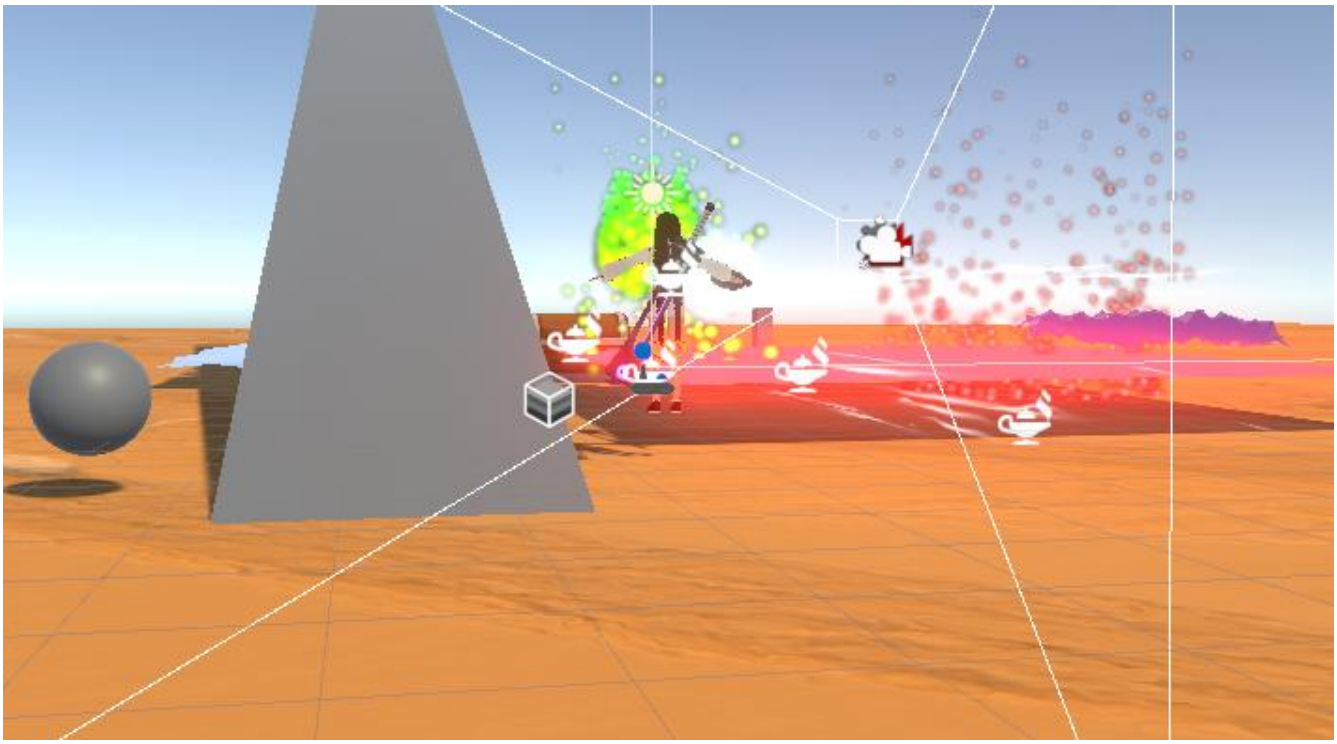

Figure 9. *The second scene.*

To build a level the author uses building blocks. Building blocks are 3D models that usually have specific height, width and depth in order to align them easily. That allows placing and combining them with great flexibility, which improves efficiency drastically. That way it is possible to create a new building in a matter of minutes or draft an outline of the future map in a matter of hours. However, one should be careful; the game level might look unnatural if a small amount of building blocks were used or if they look the same.

Figure 10. *Building blocks.*

To add variety to the exact same object, the author used various plants, stones and other decoration elements (**Figure 11**). While designing a level, it is important to keep the aesthetics intact; otherwise, the clutter of various elements might look weird.

Figure 11. *Additional decoration elements.*

Here (**Figure 12**) it is seen that two identical building blocks are aligned pixel perfect. There is no gap between them, but in spite of that it is better to place other elements between them to add some variety. It is possible to combine different objects together if they meet the criteria, i.e., proper height, width etc., not just one wall with the exact same wall.

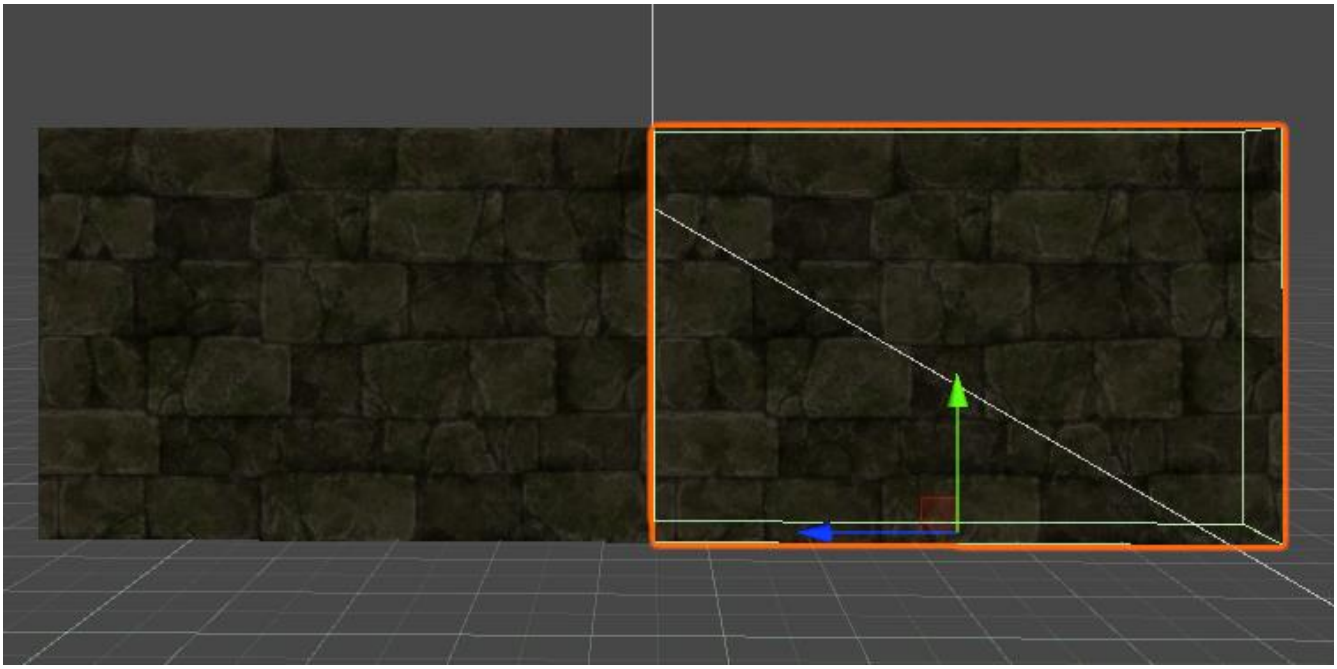

Figure 12. *Combined building blocks.*

Using blocks with the same dimensions allows merging them together with no clipping. Clipping occurs when two object intersects or one object overlaps another (**Figure 13**). Sometimes it causes weird behavior, such as glitching. This is noticeable, especially when the player is moving or rotating the camera. It is crucial to pay attention to such details, otherwise it might spoil the impression of the game.

Figure 13. *Clipping.*

Visual effect graph (**Figure 14**) allows to make complex visual effects using millions of particles because it runs on a GPU, unlike Unity's built-in particle system, that runs on a CPU. It is built through a node-based interface where no code is required, and its interface is customizable. Logic runs from top to bottom. For this project, Visual effect graph is preferable, because there is no need to interact with physics engine unlike built-in particle system that can do it with no extra amount of effort to put to make it work.

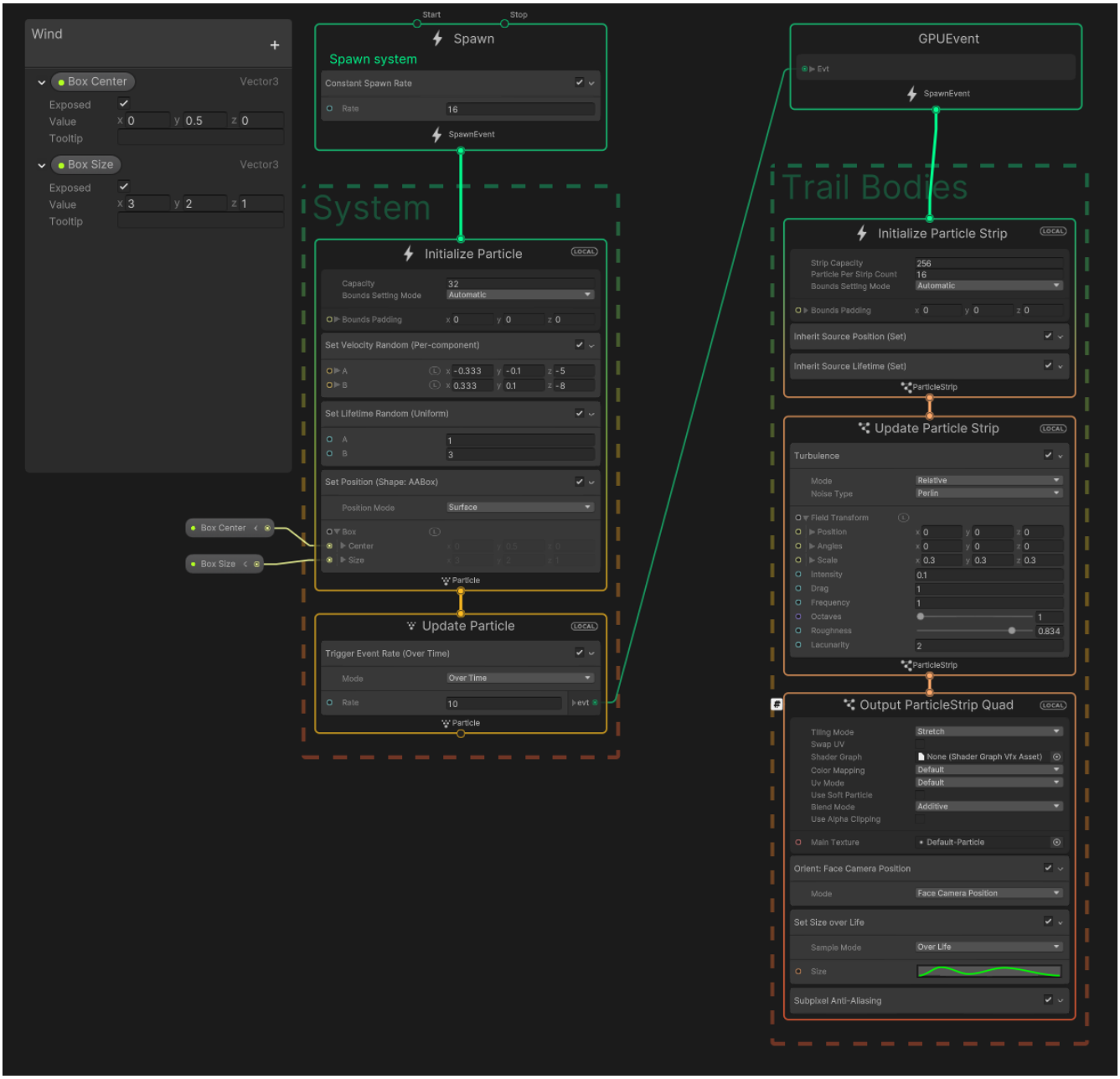

Figure 14. *VFX Graph example.*

One of the most important visual effects in the game is the fog effect. It is placed under the buildings to make an impression that the action takes place somewhere high in the sky. The player can see the fog almost all of the time and this effect is most significant for visual experience. Like other visual effects in the game, it is made using Visual Effect Graph.

Figure 15. *Fog visual effect.*

Other important elements of the game are shaders. Shaders, as well as the visual effects, have a very heavy impact on the player's experience. If the game has astonishing effects and shaders, the player may not notice small flaws in other areas of the game or the game process. Shaders are small scripts that calculate color of each pixel, based on the lighting input. They also can manipulate each pixel position and deform it, they can hide objects, make them glow and many other things. In the game, one of them is a water shader. Water shader is applied to a simple plane, deforms it, and changes its behavior.

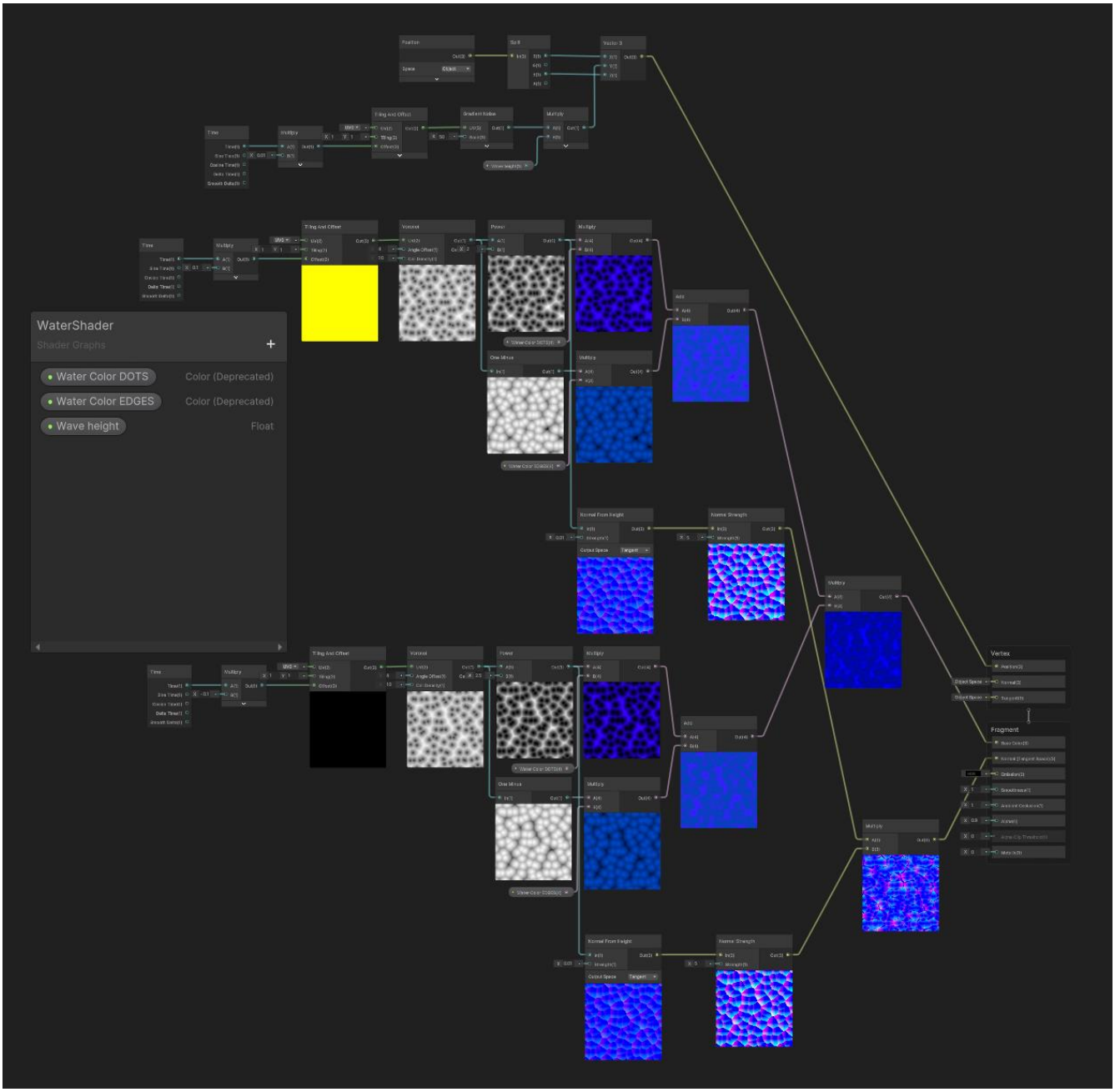

Figure 16. *Shader Graph example.*

Another big part in game development is 3D modeling. The player sees a character throughout the whole game process; hence it is necessary to make the model right. It is a whole process with many caveats, but

key parts in 3D model creation are modeling itself, texturing, rigging and weight painting. Author did all that in Blender.

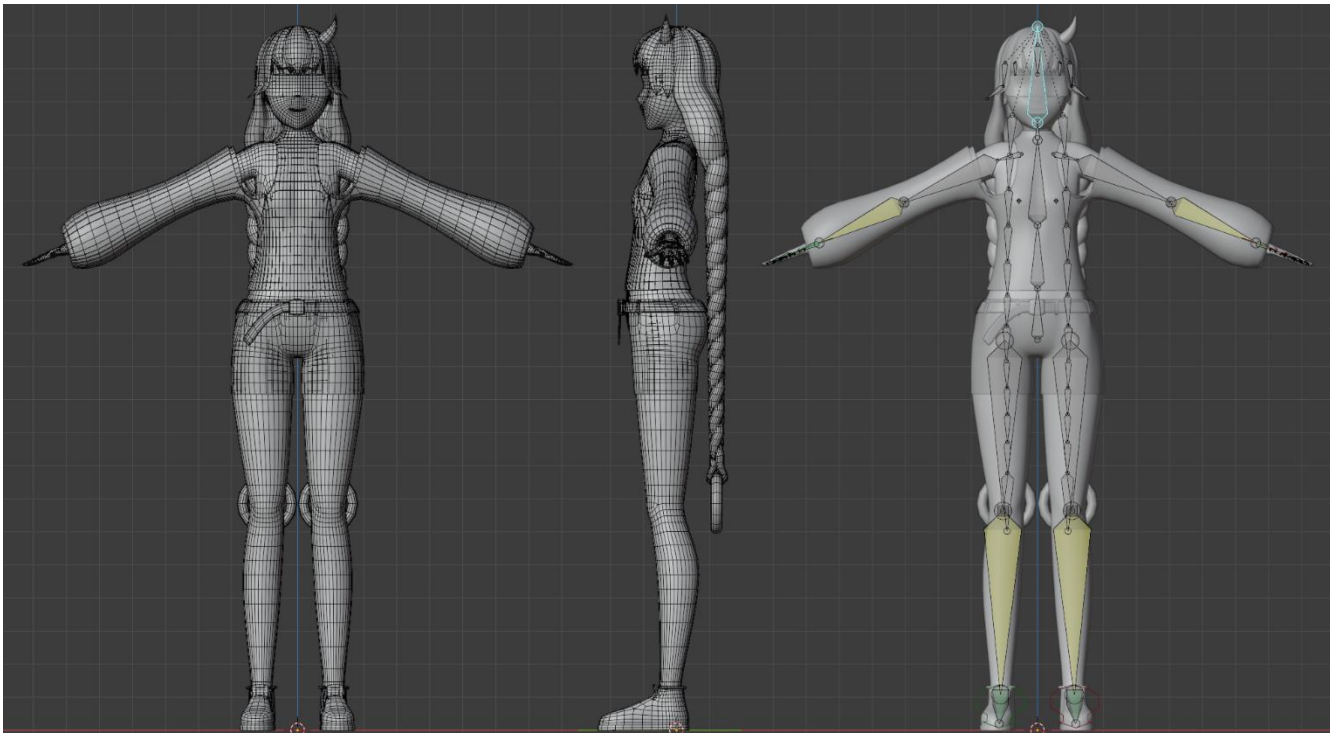

Figure 17. *Character model and its bones.*

Character animations were made in Blender because it is more convenient thanks to IK. IK simplifies animation process so one can affect desired number of bones at the same time and some of them change their position automatically based on another bone position. Animation is the process where specific model bones are positioned at desired place in desired time. Other object animations, for instance chest opening animation, were made in Unity, because they are not required such complexity, they are easy to make and that way there is no need to export animations from Blender to Unity.

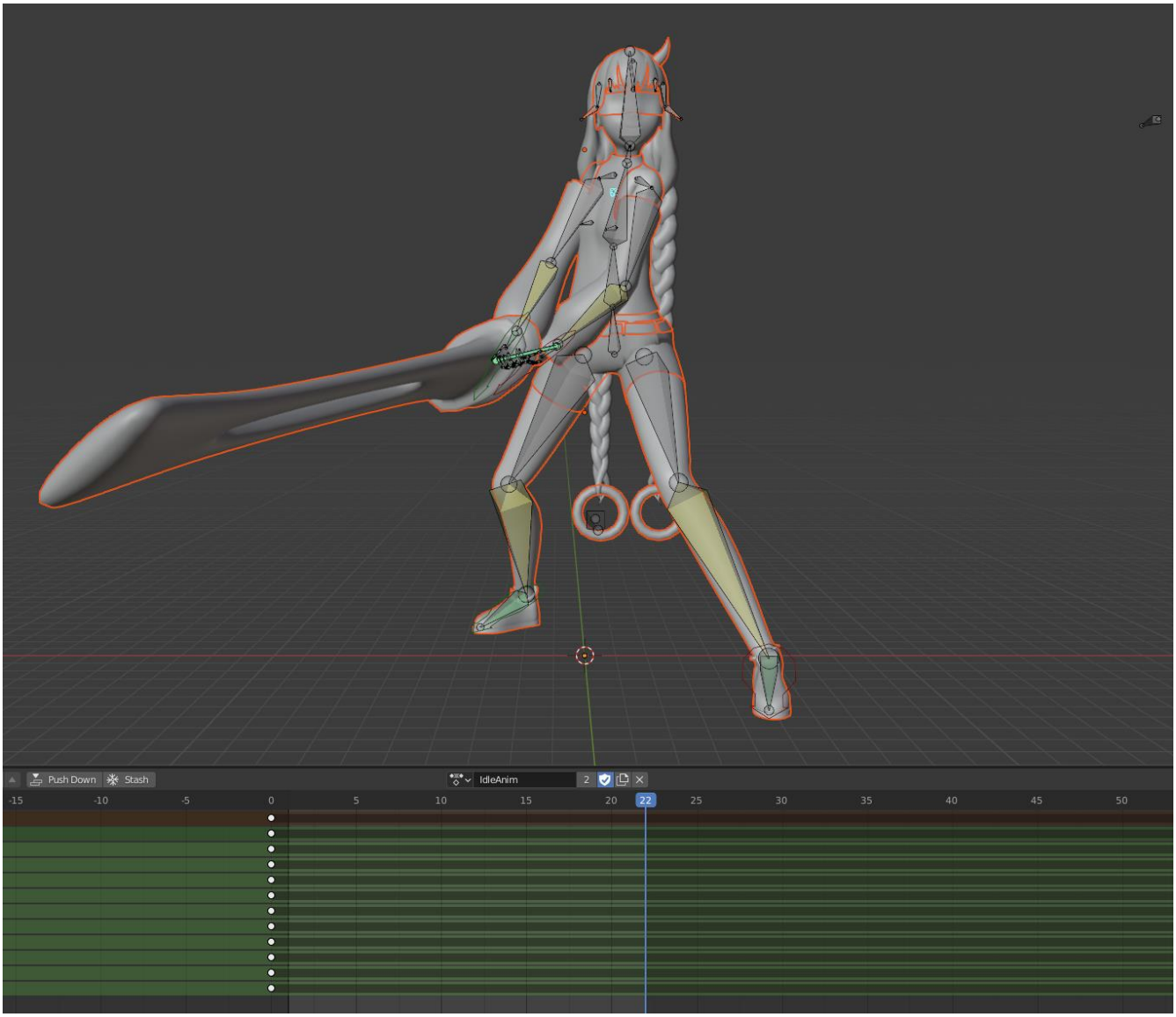

Figure 18. *Character animation process.*

When the animation process is finished, it is time to export the model itself and its animations into Unity. In order for animations to work in the Unity game engine, it is necessary to create an animator controller and a character avatar. Character avatar (**Figure 19**) is a way for Unity to configure a model's skeleton. There are two types of skeletons for the model: a humanoid and a generic. The humanoid has a specific structure, and its bones are organized in a way that somehow conforms to a human skeleton. It is crucial to make the right structure of the bones in the rigging process, otherwise it won't work right. Generic is

for anything else, be it a mutant slug or a beautiful butterfly. The main feature of the humanoid type is that its animations can be applied to another model with humanoid type, whose skeleton will have the same structure. In that case, newly added characters could reuse the same animations as the previous characters.

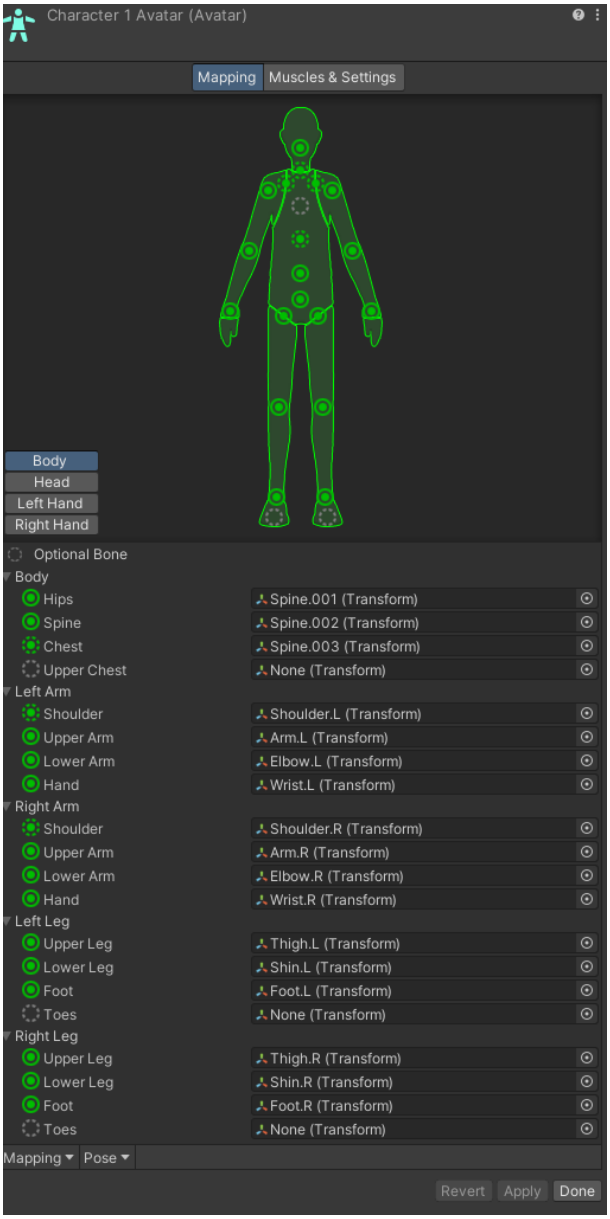

Figure 19. *Character avatar.*

Animator controller allows assign and maintain animation clips and associated transitions to the configured character avatar or any other object. In animator, transition conditions between animations are assigned. In order to transition from one animation clip to another, e.g., from fighting stance to idle, some conditions must be met. When the game is first launched, the character starts in the idle state and idle animation will be played. When the player presses some movement button, e.g., “W”, parameter from the right list (**Figure 20**) will be activated and condition for transitioning from idle state to running state will be met. Animations are blended together and transition between one animation to another usually is smooth.

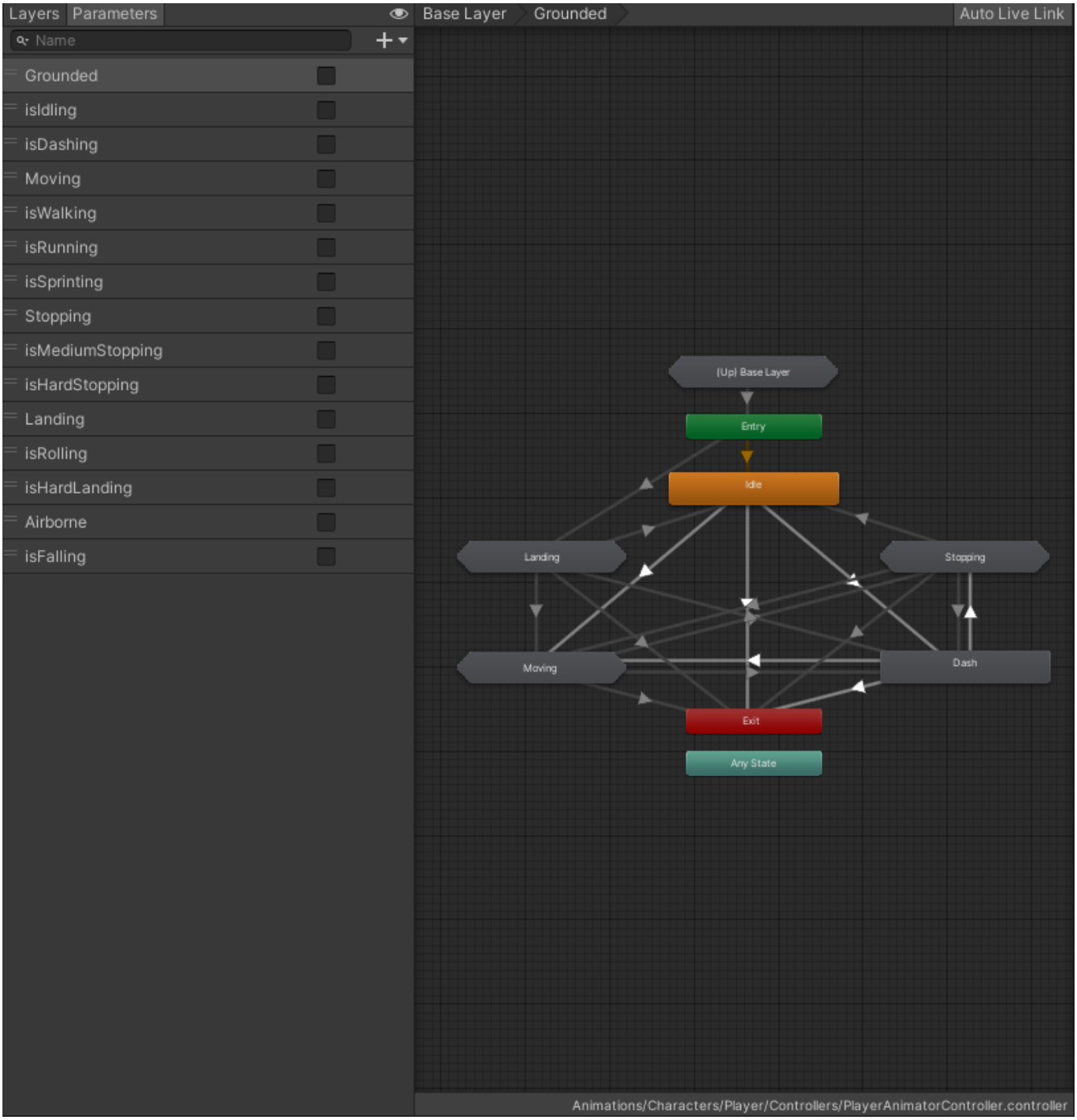

Figure 20. *Animator controller.*

Application Classes

Interactable.cs

If a game object has this script, it means that the player can interact with this game object. This object must have a collider attached in order to work properly. For instance, there is a chest before the player, and it has interactable script attached to it. If the player is in predetermined range, they can press a specific button, e.g. „E“ button on the keyboard and this will invoke an event in another script. Simultaneously a corresponding button icon will appear.

ChestController.cs

This script cannot be run without Interactable.cs. When a character is in range and the player presses „E“ button, they immediately invoke a method from this script that allows to open a chest. ChestController script runs the chest opening animation, shader effect (dissolve effect in this particular case) and visual effects while the chest is opening.

LookDotProduct.cs

This script determines if the character is looking at the object with this script attached.

PillarSpawner.cs

This script spawn identical objects (pillars) in the required quantity and then mini game starts. In this game the player has to memorize a sequence, in which objects will be highlighted, and then repeat it. If the player was able to repeat the sequence, the game would end, and the player would be given a reward. If the player fails to repeat, the mini game will end, and the player will be given the opportunity to try again.

BridgePuzzle.cs

This script cannot be run without Interactable.cs. When the character is in range and the player presses „E“ button, they immediately invoke a method from this script that lifts a bridge out of the water, and then the player can cross the bridge.

FlyingPlatform.cs

This script makes a platform move when the player stays on it for a certain amount of time. The platform starts to move to the closest target point that was placed somewhere in the scene. When the platform reaches the closest target point, it will wait for a certain amount of time once again and starts to move to the next target point and so on. When the platform reaches its last destination, it will wait and then go backwards, so it will return to its initial position.

MoveBlockPuzzle_ChildTrigger.cs

This script checks if the player intersects object trigger/collider. If they do, then the script will execute method from MoveBlickPuzzleController class. When the player exits the trigger/collider then the script will execute another method from MoveBlickPuzzleController class.

OpenDoor.cs

This script cannot be run without Interactable.cs. When the character is in range and the player presses „E“ button, they immediately invoke a method from this script that allows a door to open.

PlatformAirRoad.cs

Same as the FlyingPlatform.cs, it makes platforms move when the player stays on it for a certain amount of time. The only difference is that all the platforms start to move at the same time in order to form a road that the player can walk on.

Platform.cs

This script changes the interpolation mode of the character’s rigidbody so there is no camera jittering when the player is moving on platform.

RespawnPoints.cs

This script contains all the saving point locations, so when the player falls down, they will be transferred to the last activated saving point location.

PlayerFallDown.cs

This script checks if the player enters object's trigger/collider. If they do, then it will execute method from RespawnPoints.cs that allows the player to resurrect.

PlayerReachedSaveZone.cs

This script checks if the player enters object's trigger/collider. If they do, then it will run VFX, disable a trigger and then execute method from RespawnPoints.cs that will enable next saving point.

SwordPrize.cs

This script cannot be run without Interactable.cs. When the character is in range and the player presses „E" button, they immediately invoke a method from this script that allows to pick up a sword from the ground. At the same time the shader effect (dissolve shader in this particular case) is running.

WindBlow.cs

This script checks if the player enters object's trigger/collider. If it does, then it will add force to the player, and they will be thrown higher in the air.

Puzzle_Airflow.cs

This script checks if the player stays in object's trigger/collider. If they do, then it will add force to the player, and they will be pushed back little by little. Player might disable it by using the lever.

Puzzle_RP_ChildCollider.cs

This script checks if the player intersects object's trigger/collider. If they do, then the object will change a material, e.g., from red to blue. Then the script will execute method from RepeatPattern class.

RepeatPattern.cs

This script checks if the player pressed "buttons" in the right (predetermined) sequence. If the sequence was right, then the script will execute the method from OpenDoor.cs. If the sequence was wrong, the player has the opportunity to reset already pressed buttons and reset the puzzle to its initial state.

MoveBlockPuzzleController.cs

This script checks if the block intersects object's trigger/collider. If it does, then the object will check if the required amount of blocks are inside. If the requirement were met, the doors would be opened.

CameraZoom.cs

This script allows the player to zoom camera in and out at a certain speed between two predetermined minimum and maximum values.

PickUpObject.cs

This script allows the character to grab some game objects (only blocks from cube puzzle for now).

Player movement related scripts:

- PlayerMovementStateMachine.cs

This script cashes all the player movement states. In that case, the author does not have to instantiate a new player state, e.g., when changing state from idle to run every time it is needed.

- IState.cs
- StateMachine.cs
- CapsuleColliderUtility.cs
- PlayerInput.cs
- PlayerMovementState.cs
- PlayerGroundedState.cs
- PlayerDashingState.cs
- PlayerIdlingState.cs
- PlayerLandingState.cs
- PlayerLightLandingState.cs
- PlayerHardLandingState.cs
- PlayerRollingState.cs
- PlayerMovingState.cs
- PlayerWalkingState.cs
- PlayerRunningState.cs
- PlayerSprintingState.cs
- PlayerStoppingState.cs
- PlayerLightStoppingState.cs
- PlayerMediumStoppingState.cs
- PlayerHardStoppingState.cs
- PlayerAirborneState.cs
- PlayerFallingState.cs
- PlayerJumpingState.cs

Moreover, its corresponding Data scripts, that contains information such as speed modifiers etc.

Those scripts allow the character to move, change its states, e.g., from running to dashing, from dashing to sprinting, from sprinting to hard stopping and so on. In addition, they allow playing character animations. When the character enters a new state, a corresponding animation clip is played.

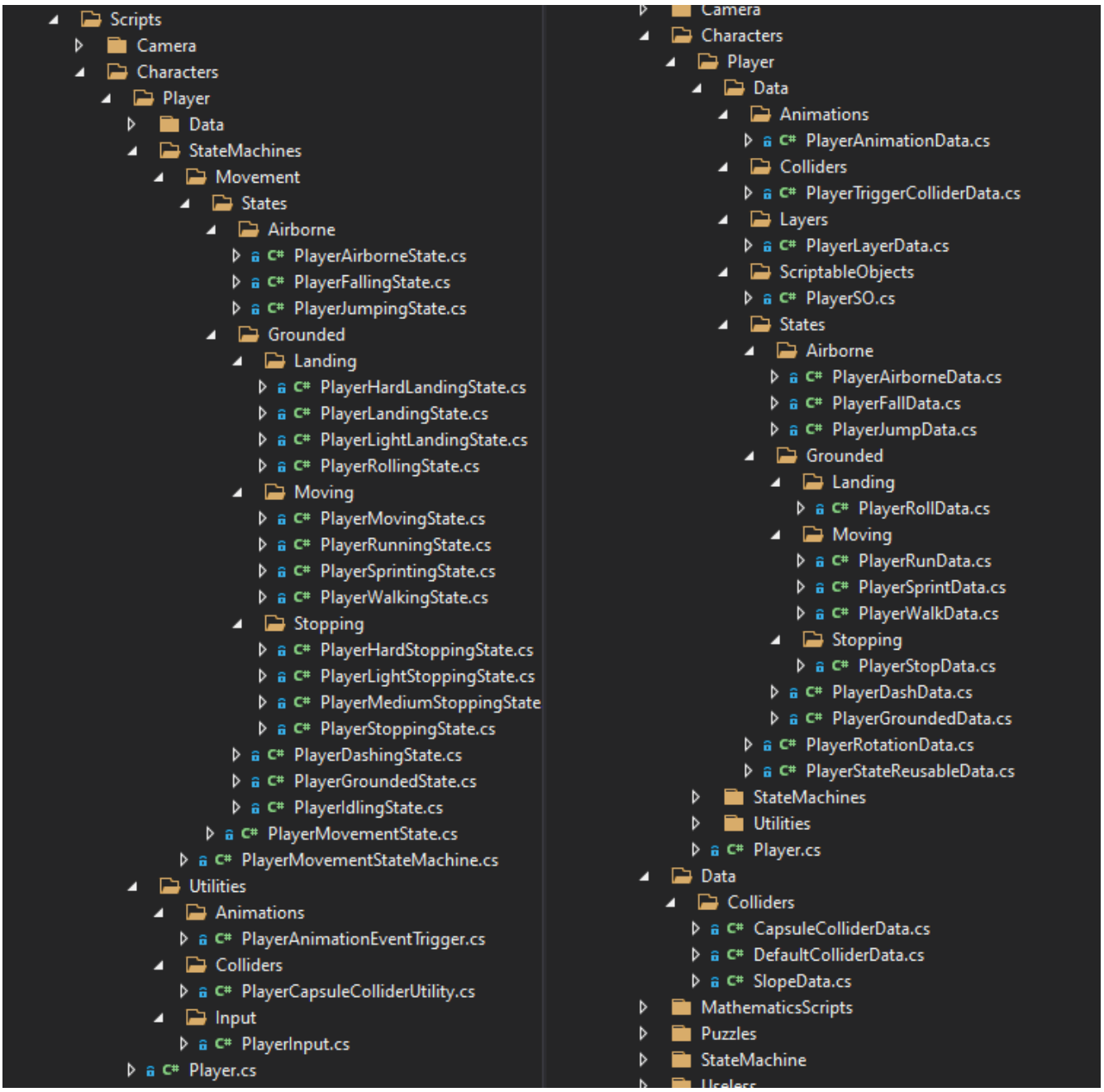

Figure 21. *Character movement related scripts.*

The diagram (**Figure 22**) shows how movement states look like. It contains all the player states.

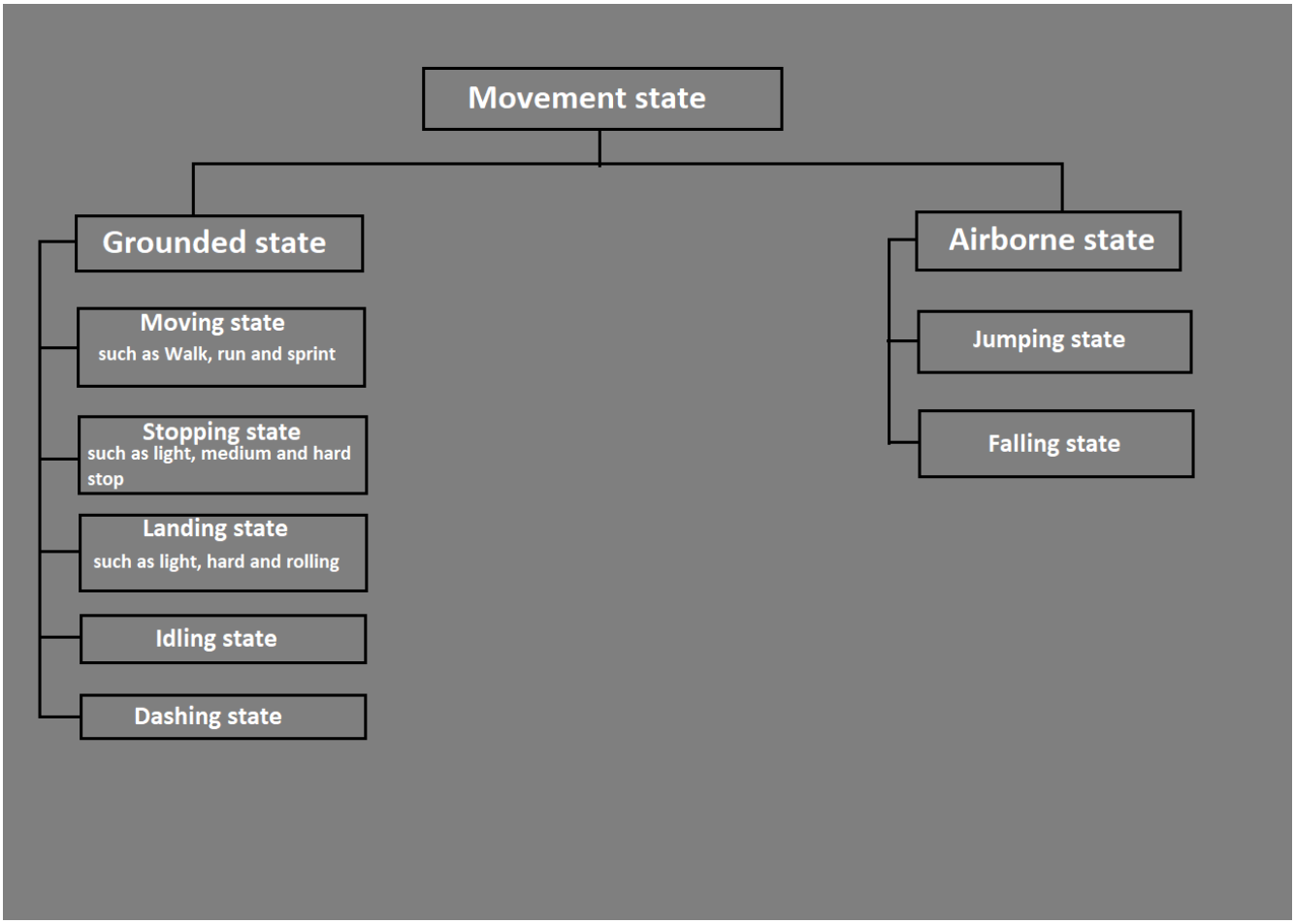

Figure 22. *All character movement states.*

System requirements

The game cannot be run on just any computer, as low performance may spoil the impression of the game. Game developers usually specify the minimum system requirements on which the game will at least run. The author cannot name specific recommended system requirements for smooth gameplay, as the author does not have access to a large amount of hardware. The game was developed on a computer with an Intel i7-8700k processor and a Nvidia RTX 3060 graphics card, 16 GB DDR4 RAM. Additionally, the game was tested on a computer with an Intel i5-4460 processor, Nvidia GTX 1070 graphics card and 16 GB DDR3 RAM and the gameplay was smooth. On a computer with an Intel i7-6700 processor, Radeon R9 280x graphics card and 8 GB DDR4 RAM, the gameplay was less stable, but quite playable.

Therefore, it would be reasonable to say that the minimum system requirements are Radeon R9 280x or Nvidia GTX 770 graphics card, Intel i5-4460 or AMD FX-8320 processor and 8 GB DDR3 RAM.

Questionnaire

After the prototype was ready, a questionnaire was conducted in which several people took part. The following questions were asked:

Q1 (What is your gender?)

Q2 (What is your age?)

Q3 (Time required to complete the game?)

Q4 (Were you able to complete the game?)

Q5 (Did you like the game?)

Q6 (Did you manage to complete the game on your own or did you need outside help?)

Q7 (Should more hints be added to the game?)

Q8 (Which puzzle did you like the most?)

Q9 (Which puzzle did you like the least?)

Q10 (Was it interesting to play the game or did you have a desire to quit halfway through?)

Q11 (Do you think that after completing this game you will feel more confident in your skills if you face similar puzzles?)

Subject #	Q1	Q2	Q3	Q4	Q5	Q6	Q7	Q8	Q9	Q10	Q11
Subject 1	Male	Under 16	10 min	Yes	Yes	No help required	Yes	Last one with cubes	Bridge search	I didn't want to quit the game	Yes
Subject 2	Female	Under 50	22 min	Yes	Yes	Help was needed	Yes	Bridge search	Cube puzzle	It was interesting	Yes
Subject 3	Male	Under 50	17 min	Yes	Yes	No help required	Yes	Cubes	Color matching puzzle	I didn't want to quit the game	Yes
Subject 4	Male	Under 16	9 min	Yes	Yes	No help required	No	Bridge search	Cubes	No desire to quit halfway through	Yes

Table 1. *Questionnaire.*

Although a small number of people took part in the questionnaire (**Table 1**), the results show that young people go through the game much faster than adults. This may be due to the fact that adults use the computer mainly for work and are not used to the controls in games. The average time to complete the game is approximately 15 minutes. All respondents were able to complete the game to the end, the majority did not need outside help. The results for the puzzle questions are mixed, there is no particular puzzle that everyone doesn't like. It is also clear from the results of the questionnaire that no one wanted to quit the game halfway through, and all respondents believe that they will feel more confident if they encounter similar puzzles in the future.

Future development

In this chapter, the author will describe features and changes that may be added in the future. The author plans to continue developing the application after defending his thesis, so he considers this chapter to be very important.

At this stage, the game lacks replay value. It might be fun to play for the first time, but it is not amusing to play the same level over and over again. It would be good to have more levels.

A reward for beating the first level is a sword. It is only natural that the next update should contain the ability to fight monsters. As for today, there are some attack animations, monster and even battle system prototype that the author created, but it was decided to exclude them from the final prototype.

With that in mind, the player has to have HP bar so monsters can defeat them and vice versa. In addition, it would be appealing to add skills (abilities) to the character in order to make the game more fun.

Game settings. At this moment, there is no way of changing game settings. It might be wise to add the ability to change in-game settings if the player PC cannot run the game smoothly.

Reward system. The game has the opportunity to loot chests, but there is no actual reward for doing so. It might be possible to add inventory mechanics and put armor/weapons to the chests.

The game has only one character. It is possible that some players will want to have a new character of the opposite gender. In the future the author might consider this idea.

Story. The game starts immediately; there is no explanation why bother completing the game level or why buildings are floating. Average games nowadays have storylines with proper explanation of the events that makes game process more fun. A plan is to come up with at least a simple storyline.

Sound. There is no sound whatsoever and some might find it irritating because most games have it. In the future the author might consider the idea of adding sound to make the game process more interesting.

Moreover, of course, every game developer wants to see his project blooming and making profit. So, the author might consider the idea of publishing the game in game services, such as Steam or Epic Games Store.

Summary

The author has achieved the desired results; most of the desired functions are working. In order to go to the next step, the author has to make some revision, changes and fixes, which was described in the Future development section. New key features must be implemented:

- New game levels.
- Battle system.
- Sound effects.
- Settings menu to change game settings.
- Reward for opening chests.

However, the application and the rest of the system are usable, and the author believes that he has achieved a good result. The author plans to continue developing the game in the future.

The author believes that this work gave him a lot of experience in the game development field. He has a clearer vision now of how this field operates.

References

[1] Wikipedia, "Unity (game engine)," 21 2 2021. [Online]. Available: https://en.wikipedia.org/wiki/Unity_(game_engine). [Accessed 7 3 2021].

[2] The Blender Foundation, "About," 10 4 2023. [Online]. Available: https://www.blender.org/about/. [Accessed 7 3 2021].

[3] Microsoft, "A tour of the C# language," 28 1 2021. [Online]. Available: https://docs.microsoft.com/en-us/dotnet/csharp/tour-of-csharp/. [Accessed 7 3 2021].

[4] Wikipedia, "Paint.net," 26 1 2021. [Online]. Available: https://en.wikipedia.org/wiki/Paint.net. [Accessed 7 3 2021].

[5] Wikipedia, "High-Level Shading Language," 26 2 2021. [Online]. Available: https://en.wikipedia.org/wiki/High-Level_Shading_Language. [Accessed 8 3 2021].

[6] Unity Technologies, "Visual Effect Graph," 2 3 2021. [Online]. Available: https://docs.unity3d.com/Manual/com.unity.visualeffectgraph.html?_ga=2.97167327.1183806980.1615125408-735636232.1593987552. [Accessed 8 3 2021].

Lihtlitsents lõputöö reprodutseerimiseks ja üldsusele kättesaadavaks tegemiseks

Mina, Anton Laagus

1. annan Tartu Ülikoolile tasuta loa (lihtlitsentsi) minu loodud teose "A study of the use of 3d interactive game solutions for logic development for school students", mille juhendaja on Dr. Chen-Wan Lin, reprodutseerimiseks eesmärgiga seda säilitada, sealhulgas lisada digitaalarhiivi DSpace kuni autoriõiguse kehtivuse lõppemiseni.

2. Annan Tartu Ülikoolile loa teha punktis 1 nimetatud teos üldsusele kättesaadavaks Tartu Ülikooli veebikeskkonna, sealhulgas digitaalarhiivi DSpace kaudu Creative Commonsi litsentsiga CC BY NC ND 4.0, mis lubab autorile viidates teost reprodutseerida, levitada ja üldsusele suunata ning keelab luua tuletatud teost ja kasutada teost ärieesmärgil, kuni autoriõiguse kehtivuse lõppemiseni.

3. Olen teadlik, et punktides 1 ja 2 nimetatud õigused jäävad alles ka autorile.

4. Kinnitan, et lihtlitsentsi andmisega ei riku ma teiste isikute intellektuaalomandi ega isikuandmete kaitse õigusaktidest tulenevaid õigusi.

Anton Laagus

9.05.2023