Creating 3D games with Unity3D

1st Boris Fuchs
Mobile Computing
University of Applied Sciences Upper Austria
Hagenberg, Austria
boris.fuchs@students.fh-hagenberg.at

2nd David Mitterlehner
Mobile Computing
University of Applied Sciences Upper Austria
Hagenberg, Austria
david.mitterlehner@students.fh-hagenberg.at

3rd Anika Seibezeder
Mobile Computing
University of Applied Sciences Upper Austria
Hagenberg, Austria
anika.seibezeder@students.fh-hagenberg.at

***Abstract*—The goal of this paper is to demonstrate the simplicity of Unity3D by creating a small 3D game. The focus is not only on scripting, but also on the features of the graphical editor provided by Unity3D. To keep costs low, only freely available assets are used.**

***Index Terms*—Unity3D, 3D Games, 3D Environment, Character Controls, Animations**

I. Introduction

Nowadays creating realistic 3D games is getting a lot easier, due to various tools like Unity3D or the Unreal Engine. Game developers need little to no programming skills to make games in a minimum amount of time, because a great amount of premade assets and scripts are available for a very low cost. Most of the developing work can be done in the graphical editor, like creating 3D figures and adding properties, which can include self made scripts or physical properties. The majority of the existing game editors provide developing on multiple platforms including Smartphones and the most common game consoles. However Unity3D is better suited for devices with low end graphics cards, because it is more lightweight. Compared to the others, the available assets of Unity3D are not as high quality, but more economical on memory and graphics.

The main point of this paper is: outlining the workflow of the game development process in Unity3D which consists of creating an environment and making a Third-Person-Character and some enemies.
The first point of the game development process in Unity3D is to create an environment, with a small forest, some mountains and a few cottages. Then some textures are added to the components. Secondly the Third-Person-Character is linked with a script, so it can move around in the environment. The third point is to create some enemies, that are chasing the player. Lastly animations for both, the character and the enemies, are provided and added to the game.

II. Creating an Environment (Written by Boris Fuchs)

The following section focus at creating a simple environment with Unity3D. Unity3D provides an asset store, where all necessary assets (environment, scripts, characters,...) can be obtained. Anyone can offer his asset-packs in the store and other users can buy this asset-pack then. The prices of these asset-packs have a high variation, the range is from free usage to very expensive.

A. Preparing Unity3D

For building a simple environment it is required to download the basic asset-pack of Unity3D and optional some other 3D-assets like trees or stones. This short environment demo is restricted to the Unity basic asset pack. The basic asset pack can be found in the store under *Unity Essentials* $\rightarrow$ *Asset Packs* $\rightarrow$ *Standard Assets*. After this step is finished, the assets can be involved in the current project or in that project where these assets be needed.

B. Environment: First step

Now its time to build a first terrain, the foundation of the prospective 3D-Models. A scale of 150 to 150 is an adequate base to represent the specified environment. As on fig.1 shown, the user can modify the scale at the last button in the terrain info template. All necessary (for this project) modifications can be done at the terrain info template. Take a look at the tabs that are interesting for crating a small environment. The first index tab is responsible for raise/lower the terrain. (e.g. for building mountains) The next two tabs can be ignored for this task. The fourth tab is for painting textures, and with the fifth tab (paint trees) the user is able to place trees at the terrain. The sixth tab is for painting details, more precisely for ”painting” grass. With the last tab its possible to change the settings of the terrain, as before mentioned, to modify the scale. The next steps take a closer look at the various index tabs.

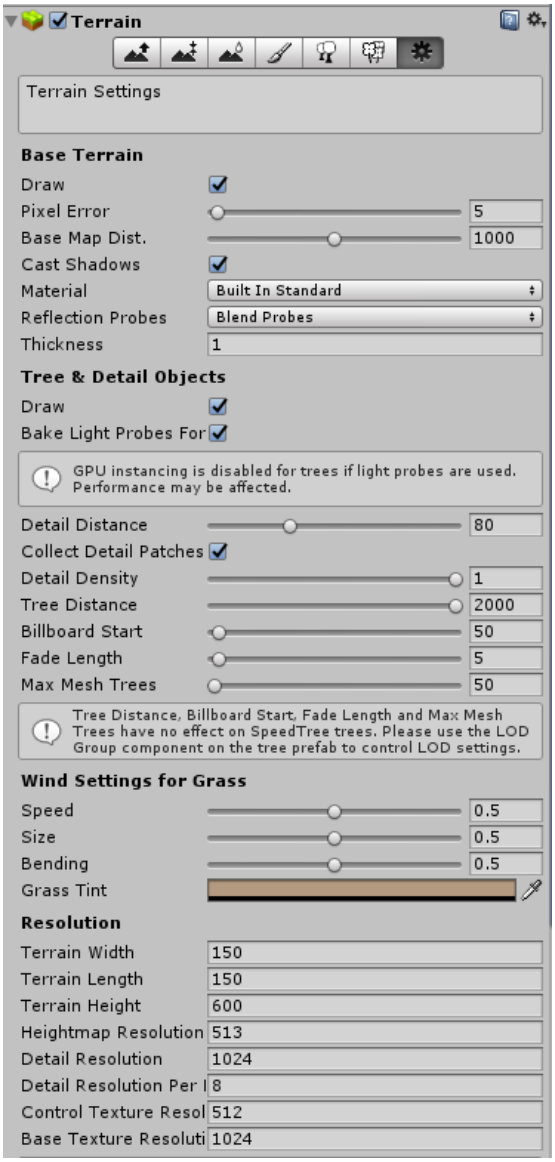

Fig. 1. Terrain Info

C. Environment: Raise/Lower Terrain

This index tab has multiple brushes for lowering or raising the terrain. (see fig.2) The best way to find out which brush is the best for altering the terrain, trying all brushes out and checking the effects will help. Additional there are settings for the brush where the user can change the brush size and the brush opacity. Depending on what the user wants to handle, these settings can be fit on every situation. To raise the terrain only click with the left mouse button and hold until the terrain is high enough. For lowering the user has to press and hold shift before clicking the left mouse button.

D. Environment: Paint Textures

One of the most important points is painting the textures. Without textures the most objects can not be identified. Like on the tab before here are also different brushes available for

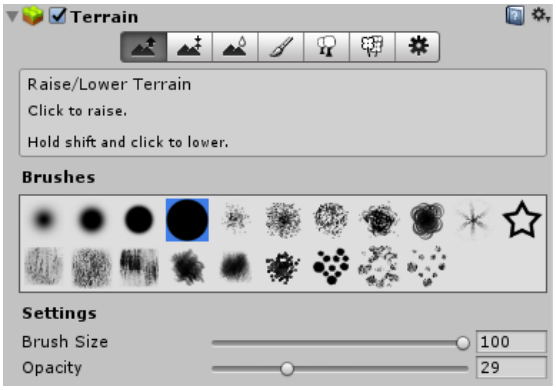

Fig. 2. Raise/Lower Terrain

selection. The next step is creating a new texture. To choose between the different textures of the Unity3D standard asset pack, the user has to click on *edit textures* → *add textures* → *select*. If everything worked, a view shows all available textures. The next two steps add the texture to the current project, first double click for adding the choose texture, then press add. Now the user can repeat this instructions for adding as much textures as want.

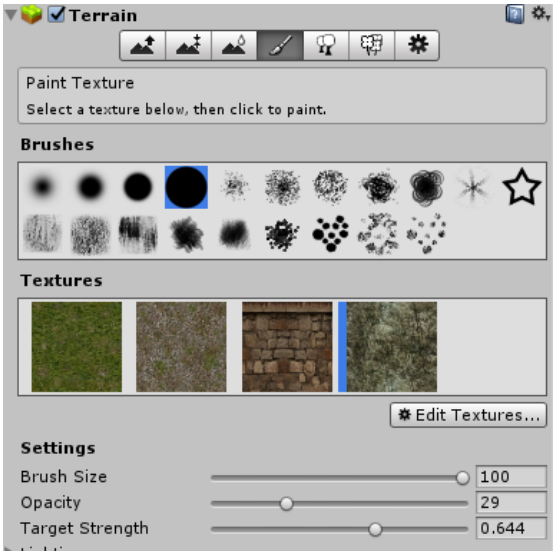

Fig. 3. Paint Textures

E. Environment: Paint Trees

Inclose varying trees to project is nearly the same procedure as discussed in the chapter ”Environment: Paint Textures”. (*edit trees* → *add trees* → *tree prefab* → *choose tree* → *double click* → *add*) Its possible to paint only one tree or to paint a lot of trees, depending on the brush size and the opacity. Now a lot of options are available. The key-features are: Adding other trees, mixing different trees and place it with a high brush size, place only a few trees or placing a dense forest depending on the current set opacity,....

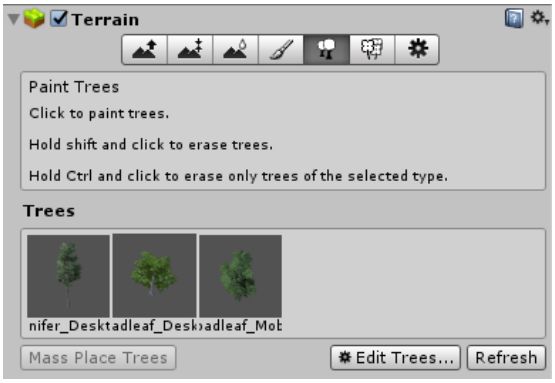

Fig. 4. Paint Trees

F. Environment: Paint Details

The last section describes how to place grass on the terrain. The method to handle that is almost the same as before. Unity3D trys to stay abreast of the same to guarantee a easy handling for users. The user has to click once again on (*edit details → add grass texture → tree prefab*). After that, a small window appears where the user can change various settings. Inter alia it contains the minimum and maximum width and height plus settings to modify the color of the grass. (color can be change later, also when the grass is already placed) Selecting a grass is possible by clicking on detail texture and then double click on the decided grass texture like before.

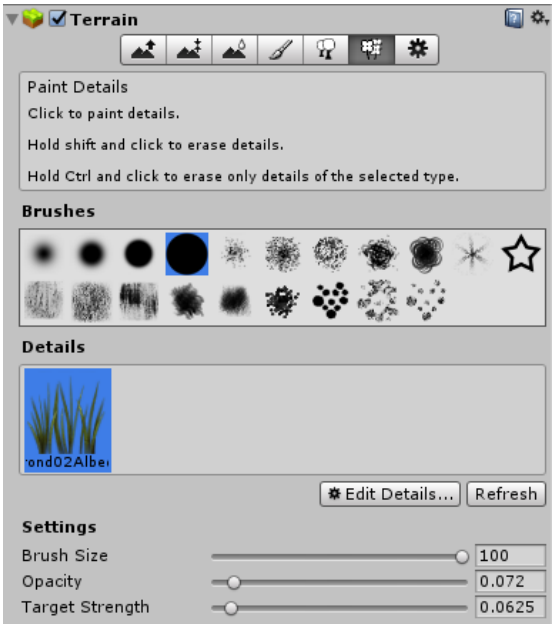

Fig. 5. Paint Details

G. Environment: Conclusion

The user is now able to deal with the basic features of Unity3D creating an environment. But Unity3D provides much more features, settings and possibilities to create beautiful environment combined with an easy and clear handling. This short description illuminated only the recommended points to create a simple terrain with grass, trees and textures.

III. Character Controls (Written by Anika Seibezeder)

Character controls are a must have when developing a game. Otherwise the user would not be able to interact with a given character. No matter how hard the user is pressing a key or moving the mouse, the character still would not move at all. With Unity3D the user interaction can be easily made by implementing scripts and adding them to the editor via Drag and Drop. The scripts can be written in C#, JavaScript and Boo [1]. In the editor itself there is no opportunity to write the code, but there are several supported editors such as the recommended editor MonoDevelop by Unity3D or VisualStudio by Microsoft [1].

A. Adding a Character

In Unity3D adding a character is easily done by dragging a Prefab or an object into the middle screen to the wanted position. There is also another possibility by dragging the Prefab or object to the left side of the screen, where all used objects are shown. A Prefab is a pre-built object combined of one to many components [2]. When downloading the Unity3D editor some prefabs are already given and can be used as is. To see and edit features of the Prefab or object, a left click on the wanted object on the left side of the screen is needed.

As seen in figure 6, a character has components like a Rigidbody, which gives the character a mass, a capsule collider, which triggers interrupts when colliding with other objects and scripts. Scripts can have some changeable variables like shown with the 2nd script, or no changeable variable.

B. Scripting in VisualStudio

A script is simply added by clicking the button ”Add Component”, choosing ”New Script” and selecting the wanted language. After adding the script it can be written by double clicking the component and opening the wanted script editor. The first thing to know about writing a script for Unity3D is, that the class is extending from the `class MonoBehaviour` and is ipmlementing the functions `void Start()`, `void Update()` and `void FixedUpdate()`. The start function is used for variable initialization and one time execution. The update function is used for continuous events, like checking which key is pressed, and is updated every frame [1]. As the user may press other keys while playing the game, the statements for checking the key should be implemented in the update function. Both functions are not necessarily needed but can help building the logic of the game. Another important function is `void FixedUpdate()`. Like the `void Update()` function, it also is updated every frame and it is mainly used when dealing with the Physics of the Object [1].
When adding variables there is just a few thing to know: all `public` variables are later shown in the component screen and can be edited there. It is also possible to import other

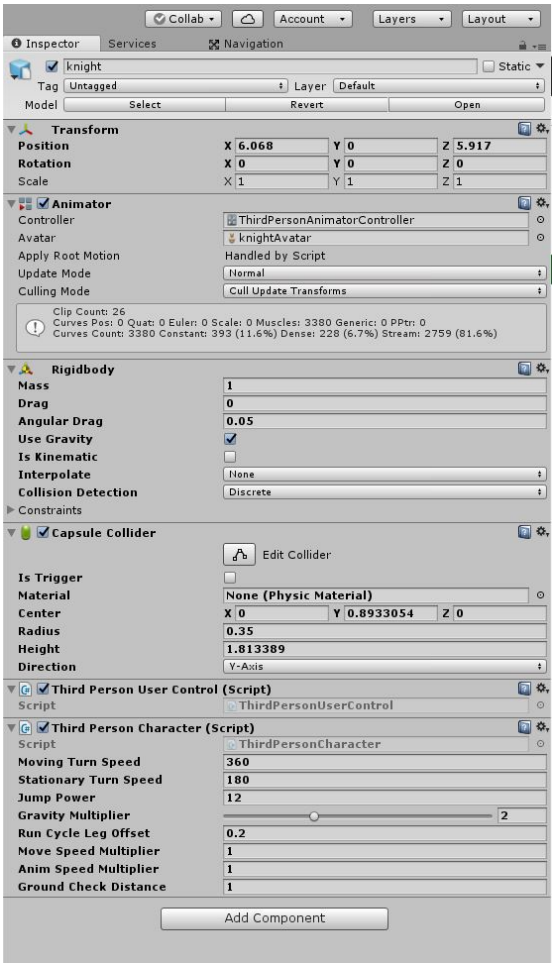

Fig. 6. Character Components

components from the object. They are added to by using `GetComponent<>()`

C. Cross Platform Input Manager

The Cross Platform Input Manager can be used to determine how keys or actions should be interpreted. For example there can be an entry considering jumping with a character, which defines, that a jump on a Desktop or Laptop is made with pressing the spacebar, while on the smartphone a touch gesture activates the jump. After defining the inputs, they can be used with `CrossPlatformInputManager.GetButtonDown("Jump")`.

Listing 1. Example Implementation for Determining a Jump

```
1 private bool isJumping;
2
3 private void Update()
4 {
5    if (!isJumping)
6    {
7       isJumping = CrossPlatformInputManager.
            GetButtonDown("Jump");
8    }
9 }
```

The code in Listing 1 shows how to implement the Cross Platform Input Manager in the `void Update()` function which is updated every frame.

IV. Adding Animations (Written by David Mitterlehner)

The Unity editor supports various ways of adding animations to an object. The object can be simple, like a sphere, or complex, like a humanoid character with multiple joints.

A. Keyframes

Animations can be implemented by setting keyframes in the ”Animation” tab. Defining new keyframes is done by clicking the record button next to the media controls and moving the time cursor to the desired frame (e.g. frame 60). Any changes made to the object in the scene, for instance scaling or rotating, will now be recorded and saved to this keyframe (60). The transitioning frames between the keyframes are generated automatically by the editor. After setting the keyframe the animation can be previewed by clicking the play button in the media controls. In figure 7 there is one starting keyframe at position 0.

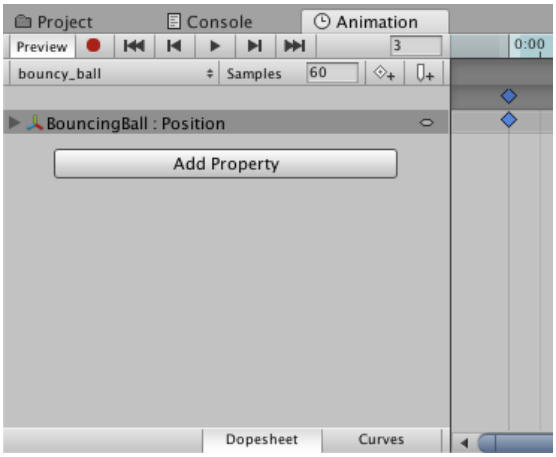

Fig. 7. Keyframes

B. State machines

Multiple animations can be added to an object using state machines. State machines allow for complex transitions and event driven animations. The Unity editor provides a colored graphical overview for managing these transitions using semantics similar to state diagrams. Figure 8 shows the graphical overview of a simple state transition between to animations.

C. Basic animation concepts

If keyframes are used for animating an object, say a bouncing ball, and no fine tuning of the duration of a single frame is performed, the animation isn’t going to look very realistic. In 1981 Disney animators published a book called ”The Illusion of Life”, outlining the key concepts to making

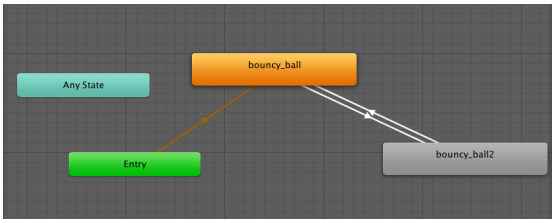

Fig. 8. Animation state machines

animations look realistic. In this publication, 12 concepts were defined, out of which two will be used here to illustrate how important fine tuning animations is for a natural look and feel of a 3D-game [3].

D. Timing

The first, and arguably the most impactful concept covered here is timing. In the bouncing ball example, a linear progression of the frames doesn't look realistic, more frames need to be assigned to the top position (slow speed), and less to the bottom position (fast speed). In Unity this can be done by switching to the "Curves" tab on the bottom of the "Animation" tab, and editing the curve accordingly (See Figure 9). The steeper the slope, the faster the ball accelerates.

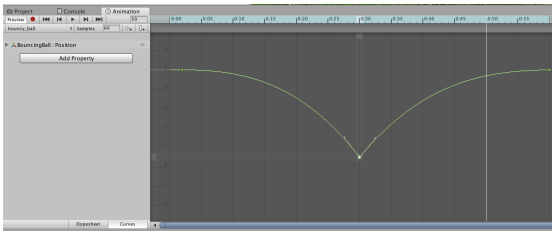

Fig. 9. Animation curves

E. Squash and Stretch

Another method for making an animated object look more realistic, is to transform the scale of the object as it moves, especially when it collides with another object. It is important that the volume of the object remains constant, however. In Unity this can easily be done by editing the X,Y, and Z scaling values of the object at a specific keyframe. Figure 10 shows an example of this.

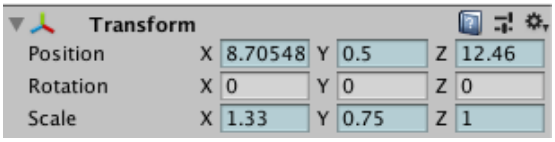

Fig. 10. Editing the scale

F. Complex character animations

The bouncing ball is a trivial example, just to demonstrate the core concepts of animating in Unity. More complex animations, like character arm and leg movements, etc. can be obtained from the asset store. For example, there is a zombie asset on the store which includes walking, idle and attacking animations. An animator is also provided, which defines when, and under which conditions the animation states should be changed (Figure 11 and 12). To make the zombie actually walk towards the player and attack it, a script needs to be attached to it, for example the "AICharacterControl" script from the standard assets, where the player (knight) can be set as the target (See Figure 13).

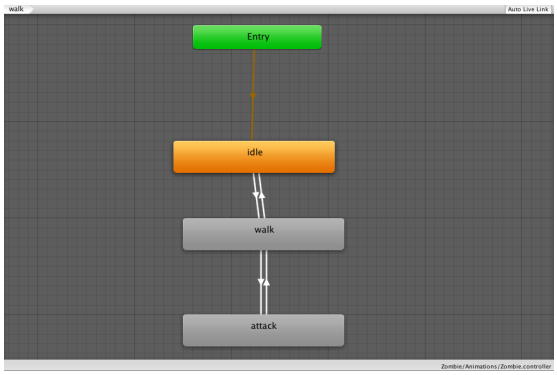

Fig. 11. State diagram for the zombie character

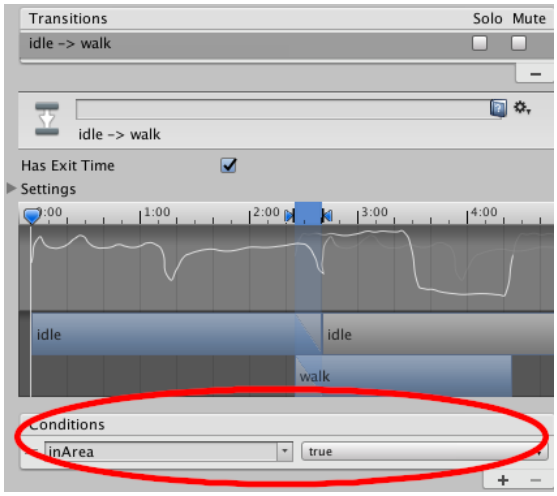

Fig. 12. Transitioning conditions for the zombie

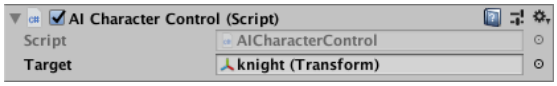

Fig. 13. Script for the zombie

REFERENCES

[1] Ji Won Oak and Jae-Hwan Bae, "Development of Smart Multiplatform Game App using UNITY3D Engine for CPR Education", vol. 9, no. 7, pp. 263–268, 2014.
[2] Unity Technologies. "Manuel:Prefabs", Publication: 2018.1-002B, 2018.
[3] Thomas, Frank and Johnston, Ollie and Rawls, Walton, "Disney Animation: The Illusion of Life", 1981, Publisher: Abbeville Press New York