Graphics and Game Technology

Assignment 6

A 2D platform game in Unity

This assignment serves as a basic introduction to a professional game engine. In this case you will use Unity to create the basis for a small 2D "platform game"[1]. To get acquainted with Unity, watch this tutorial by Unity which will introduce you to the editor. Note that the layout in your editor will most likely look different, but it will still contain the same features.

1 Setting up

The first thing you need to do is download and install Unity: go to the Unity download page and download and install the free "Personal Edition".

The framework that comes with this assignment provides some boilerplate features: a simple scene is created containing a camera and lighting. The framework also contains what Unity calls "prefabs" of a Player-object, a Platform-object and a Projectile-object. Prefabs are prototypes of objects you may want to make more than one instance of, similar to objects in Java.

1.1 Task

Load the framework project into Unity. You will see a very simple scene containing a Player (a blue square), a Platform (a green slab) and a camera (white). Extend this scene by adding more platforms to the right, adding gaps and height differences to add some challenge to, what is soon to be, the first level of your game (see Figure 1).

2 Controlling the Player

In Unity, any prefab can be extended by adding components to them from the Inspector screen. These components can be many different things: physics components that control how prefabs behave under the influence of forces, renderers that control how they are rendered, etc. but also custom scripts. By creating a

[1] Wikipedia defines a platform game as *a video game which involves guiding an avatar to jump between suspended platforms, over obstacles, or both to advance the game.*

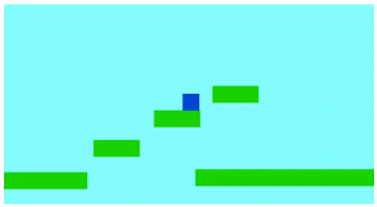

Figure 1: The initial framework, already extended with additional platforms.

new script it is possible to define custom behaviour, ranging from scripts that (for example) implement enemy intelligence ("AI") and player input scripts that keep track of lives and score. Scripts can be implemented in C# ("CSharp"), Javascript or Boo. You should use C# for this assignment because it is most similar to Java, which you already know.

When adding a new C# script, an application called "MonoDevelop" opens showing a base script containing two functions: `Start()` and `Update()`. `Start()` is a function called by Unity when initialising the object and should be used to initialise any variables you might need. `Update()` is a function called regularly by Unity. In total, there are three different update functions, all having slightly different characteristics and uses:

1. `Update()` is called irregularly, depending on framerate, and is most commonly used, mostly for input handling, simple timers, etc.

2. `FixedUpdate()` functions the same as `Update()`, but instead it is called at a fixed rate. This function should be used when working with physics.

3. `LateUpdate()` is an update function that is called after every normal `Update()` cycle. This function is used in cases where it must be assured that this action happens last, such as moving the camera based on the players position update.

Using the wrong update function can, in larger games, have unwanted effects such as choppy camera movement and physics behaviour that depends on the framerate at which the game is played (imagine a racing game where you can drive twice as fast if the framerate is twice as high).

Unity has numerous ways to obtain player input from within scripts, which can all be found here. We recommended that you use the functions `GetButtonUp()`, `GetButtonDown()`, `GetAxis()` and `GetAxisRaw()`, as these enable the end user to configure their own controls.

To implement the actual movement there are even more options you can use: you could change the absolute position of the object, translate it relative to its current position, or even add a force to the rigid body, and more. It all comes down to personal preferences and, in this case, collaboration with the input functions used.

2.1 Task

For this task, implement the following features using what is explained above:

1. Allow the player to move left using the 'a' key and right using the 'd' key, using input and movement functions of your choice. Experiment with the movement speed.

2. Allow the player to jump by pressing the 'w' key. Experiment with the height and duration of the jump.

3. Have the camera follow the player throughout the level.

Hint: For this last step, you need to have a reference to the camera from within the Player object. To do this, make the variable that is used for this reference `public` in Player. This will make the variable show up in Unitys Inspector panel, allowing you to simply drag the camera object from the Hierarchy panel into the variable in the Inspector panel.

Note: MonoDevelop, Unity's scripting editor, contains a small bug with regards to the Dutch "US International" keyboard setting. If this setting is used, apostrophes and quotes cannot be entered. The workaround is to temporarily set your keyboard settings to US English.

3 Projectiles

Next to static game objects, it is also possible to dynamically create objects using the `Instantiate()` function and to destroy these using `Destroy()`. These functions allow you to spawn and destroy objects during certain events within your game. A simple example is the use of projectiles: one can be fired (i.e. instantiated) using a button and destroyed when it touches something, along with the potential enemy it may have hit.

To detect whether an object touches another object, "colliders" are used. All prefabs in our framework come equipped with a "box collider". If two colliders touch, `OnCollisionEnter2D()` is called for both objects, after which Unity will use its physics engine to handle the further movement.

3.1 Task

For this task, implement the following features using what is explained above:

1. Implement a way by which a Projectile prefab is instantiated when the spacebar is pressed. Only one projectile can be fired at a time.

2. The projectile must be fired in the travel direction of the player. The projectile has to follow an arch trajectory.

3. The projectile gets destroyed when it hits anything or ends up outside of the visible area. Use `OnBecameInvisible()` to detect if the projectile went off-screen.

4 Triggers

The previous section briefly introduced collision detection. However, box colliders have another feature that is useful to game development: they can act as "triggers". Triggers are similar to colliders, but the objects will not physically collide. Instead, triggers merely detect whether some object has entered a specific area (encapsulated by a trigger) and call `OnTriggerEnter2D()` when this happens. Objects can have triggers and box colliders enabled at the same time. To create a trigger, add a box collider component and check the "Is Trigger" option in the Inspector.

4.1 Task

For this task, you are going to use triggers to create "one-way platforms". One-way platforms are special platforms where you can jump through the bottom and land on top, as illustrated in Figure 2. Even though Unity does not natively support them, this is not hard to create by placing a trigger below the platform. This trigger has to be slightly wider than the platform itself and has to overlap with approximately the bottom half of the platform. If the player enters the trigger, disable collisions between it and the player. Once the player exits the trigger, re-enable the collision.

Figure 2: Jumping onto a one-way platform.

5 Enemies

Nearly every video game contains some form of enemies. In this game, these enemies consist of dark cubes similar to the player, but smaller. An enemy object is very similar to a player object and can thus be used as a basis where only a few things have to be edited.

An enemy has the same physical properties as a player, albeit smaller. It can move, it has a mass. Therefore, the entire player prefab can be copied to create a new prefab called "Enemy", in which only a few things have to be edited: size, colour and behaviour.

Materials define an object's visible properties: colour, texture and bump mapping settings. In the case of our simple game, they define nothing more than the colour. Materials are assigned to the "Mesh Renderer" of the prefab, the component that controls how an object is rendered.

While the player is controlled by user input, the enemies will have to move on their own. A simple and easy to implement this behaviour is to make the enemies patrol on platforms, where enemies walk back and forth on a platform. To make sure the enemies don't walk off, they will need to know when they reach the edge upon which they turn around. In the previous task you have seen that triggers are perfect for detecting whether an object has reached a certain location.

5.1 Task

For this task, implement the following features using what is explained above:

1. Create a copy of the Player prefab and call it "Enemy". These enemies are slightly smaller than the player, so edit their size. Create a new material as a copy of the Player material and give it a dark colour. Remove the player behaviour, and create a script that makes the enemies behave as described above. Spawn enemies throughout the level.

2. Extend the Platform prefab with triggers on the edges. Use these platforms for the behaviour of enemies only. Ensure the player is not affected by these triggers.

3. Finish the behaviour of the Projectile prefab. If an Enemy touches a Player, the Player dies. If a projectile hits an enemy, the Enemy must be destroyed as well as the Projectile.

6 The User Interface

Within Unity, User Interface (UI) elements can be created in a similar fashion to other game objects. To do this, click the "Create" dropdown menu from the Hierarchy panel and select the UI submenu. When creating a UI element, a canvas is automatically generated as its parent, which is the game object that UI elements will be drawn on. This canvas supports various settings. The "Render Mode" allows you to use a camera's viewpoint as screen space. By attaching it to a camera, the UI elements drawn upon the canvas will follow the camera and remain on screen.

6.1 Task

For this task, implement the following features using what is explained above:

1. Create UI text that is shown on screen the entire time. This must contain: an arbitrary form of score (time passed, a number based on the enemies defeated, etc.) and the number of lives remaining.

2. When the player dies by an enemy or falls off the screen, remove a life and reload the level. The application defines a variable `loadedlevel`,

containing the current level. When a level is reloaded, everything but static variables get re-initialised. Use this to save some variables, such as the number of remaining lives.

3. Create a finish point for the level.

7 Bonus

You have now created a basis for a 2D platforming game. Expand on this for extra points. Possible expansions include:

- Multiple levels: Create multiple levels the player can play through. Each level must be different.
- Different lighting can do a lot to change the overall impression of a game.
- Smarter enemies: improve the AI of the enemy character. For example: Make multiple kinds of enemies, such as enemies who fly up and down. Make the walking enemy detect the players presence and make it move towards it at a faster pace.
- Add sprites and animations to the game.
- More advanced UI elements: add a high score feature, a menu at the start of the game, etc.
- Feel free to be creative and make your own expansions!

8 Grading

The grading for this assignment will be performed as follows:

- You get max 2 points for correctly working Player and camera control in a level: the Player can move left, right and jump onto other platforms, the camera follows the Player, a level contains enough platforms positioned in a manner that they form a sufficient playing challenge, there is a finish point in a level.
- You get max 2 points for correctly working Projectiles: the spacebar emits a projectile from the Player, in the travel direction and in an arch trajectory, if a projectile hits an Enemy, the Enemy as well as the Projectile gets destroyed.
- You get max 2 points for correctly working Enemies: there are several of them in each level, they look sufficiently different from Players and Projectiles, they patrol platforms, they "kill" Players when they touch them.

- You get 1 point for correctly working one-way platforms.
- You get 1 point for a correctly working User Interface: at minimum a score is shown and the number of remaining lives.
- You get max 2 points for bonus items.