Author: Nils-Inge Isaksson

Technical Documentation for Fish Defense + credits

Architecture

Unity uses a component system by default. This type of system was described in lecture 1 and means that assets such as scripts, materials and collision volumes can be attached as components game objects. This leads to a high degree of reusability since a movement script can be attached to all gameobjects which are supposed to move across the game world.

Scripts should from my perspective ideally have fairly clear and distinctive responsibilities. This is to some extent the case in my game since I for example do have a mover script which is responsible for making enemies move across the water towards the fish resource. I also have a PlacementController script as well as a projectile script which also have fairly well defined and specific responsibilities. There are however some instances where some of the scripts ended up a bit too broad in terms of responsibility. This is particularly the case regarding the Tower-script which handles a large number of behaviours in regards to towers. Some of them should perhaps have been separated into their own script. Having a separate Attacker script for example would have made a lot of sense, since it could have been applied to ships as well.

I do use a GameController which is a type of entity that is described in Lecture 1. It is mentioned that this type of controller generally is responsible for tasks that the player cannot directly influence. In my game the GameController is responsible for defining the win and loss conditions and whether any of these conditions have been met. It also handles some general counting procedures as well as changes to the game world. There is also a PlayerController which is responsible for handling tower selection, as well as bombardment (when the player manually attacks ships by clicking on them). The PlayerController is not responsible for managing all direct player derived actions. I thought it was reasonable to have a specialized PlacementController for placing tower when the player clicks on tower foundation since it is such a important action. Tower upgrades and repairs are handled by the Tower script.

Graphics

I haven't put much effort towards graphics in my project. I simply used the standard shaders that were included with the materials which I imported or which I created. I did occasionally adjust the "Metallic" and the "Smoothness" properties of the main maps, since I wanted a slight metallic and sharp style for my islands.

All the materials which I made by myself are quite simple since I just selected a desirable colour and adjusted the aforementioned properties of the shader. I simply imported more complex materials.

I barely used any lightning beyond the default for my first level. I wanted this level to be simple, warm and colourful and this was generally already the case when using the default directional light together with the existing materials. The directional light is angled straight down towards the surface of the water. Since the light is fairly neutral (I changed the tint slightly) it is reminiscent of the sun during daytime, contributing to the colourful and light hearted style of this level.

I spent more time working with lightning when I crafted the second level. I wanted to create the feeling that the sun was going down late in the afternoon, so I gave the directional light a darker reddish hue. This made the scene darker, while also giving a red tint to the materials on the surface.

Since the scene was somewhat dark I decided to use point lights to illuminate a number of tower spots on the map to make them feel more distinctive from the surrounding game objects. According to lecture 5, a point light has a position, intensity and colour. I adjusted the position to be slightly above the tower spots. When a tower is spawned at a spot the light also partially illuminates the tower. I had to adjust the intensity so the spot wouldn't be bathed in light by adjusting the distance of the point towards the tower spots, the range property as well as the intensity property. I gave the point lights a more yellow colour to make them a bit richer.

Collisions and Geometry

Collision detection is handled through colliders. Each tower and enemy unit has a capsule collider attached as a component, while each projectile which the tower or enemy shoots also has collider of its own (often a sphere collider). If a projectile enters a foreign collider which does not belong to its parent nor the tag associated with the parent, the gameobject which owns the foreign collider will register the damage caused.

This particular solution to collision detection would cause significant performance issues for a game with a large number of gameobjects. It does however work well in this particular context. Since there aren't a huge amount of gameobjects with colliders on the map, the collision detection system is not a huge performance bottleneck. Of course, it is important to systematically remove gameobjects which are no longer relevant, such as projectiles which have hit their enemies or enemy boats which have been destroyed or have managed to reach their goal. The colliders are also rather simple from a geometrical perspective, which should also reduce substantially the performance requirements as mentioned in the third lecture. This works quite well in a tower defense game since having very complex mesh colliders wouldn't improve the game.

All the towers and enemies have a script called "Damage detection" attached as a component. The script is responsible for detecting the presence of a projectile using the OnTriggerEnter event.

Each enemy ship has a rigid body attached to it. The mover script which is also attached to each enemy applies force to the rigid body, which makes the enemy travel in a specific direction.

Design Patterns and Data Structures

Towers and enemy ships use arrays of gameobjects in order to keep track of valid targets. The scripts for the enemies and the towers simple loops through all the targets and finds the one which is closest. That gameobject will then become the target. Each time this type of check is being performed the array is reinitialized. Since no elements are being added or removed after each initialization, arrays are suitable for storing this data.

However, this solution would have been quite poor had the game had a larger scope. If there had been say 3000 enemy ships on the map, then it would be important to have a single general List or Queue of towers which all the ships would have been able to access.

The WaveController which is responsible for unleashing waves of enemies uses an array of Wave-objects. Each wave-instance corresponds to an enemy wave. The Wave-class only contains information in regards to the number of enemy units (for example: public int lightships). The Wavecontroller then spawns the

specified number of ships in each wave. That makes it extremely easy to specify the exact composition of the waves directly in the inspector. Since no elements will be added or removed from the waves after the game starts, the array is a suitable data structure since they have a fixed size as mentioned in Lecture 1. This solutions was influenced by the tutorial video in **resource A**, but the difference is that I adjusted the solution so that I would get direct control of each single wave from the inspector. I desired a very high degree of control over the individual waves since I wanted to add bosses at the very end of the waves.

Finally, I used a List to keep track of the fish objects. A total of six fishes reside at the end of the game area. Whenever an enemy boat manages to reach this area, two of the fishes in the list are removed from the List and destroyed. Since the list automatically adapts to this removal process, I can always tell the list to remove the first element [0] in it.

Animations and Rotations

The head parts of the towers can rotate towards the enemies. Firstly the direction of the enemy is stored as a 3-dimensional vector. Using the LookRotation method for the Quaternion class I was then able to find the proper rotation. I then applied this rotation to the head of the tower, which were easy to find since I had tagged all the head parts with "Head". The rotation is represented as a Euler Angle but is converted to a Quaternion using the Quaternion.Euler() method. According to Unity, this avoids the problem with Gimbal Lock which was mentioned during the fifth lecture.

I have not produced any animations on my own. I have only imported a single animation from the asset store which was bundled with the Crucian Carp fish model.

Tutorial Resources I have Used during the Development Process (beyond the lectures and the official Unity docs):

Glaude, M. (2016). *1 Hour Programming: A Tower Defense game in Unity 3d [Tutorial]*. Retrieved from: **https://www.youtube.com/watch?v=b7DZo4jA3Jo**

Gershman, D. (2016). *[Unity3D] Tower Defense Game Tutorial Series.* Retrieved from: https://www.youtube.com/playlist?list=PLh2Z6Jn1TBFbymvDUS8JxRwYKE0Yxfqvi

Assets used:

Boats: https://www.assetstore.unity3d.com/en/#!/content/61369

Materials: https://www.assetstore.unity3d.com/en/#!/content/63714

Stone Tower: https://www.assetstore.unity3d.com/en/#!/content/50215

Turrets: https://www.assetstore.unity3d.com/en/#!/content/9872

Fish: https://www.assetstore.unity3d.com/en/#!/content/46132

Simple Particles: https://www.assetstore.unity3d.com/en/#!/content/3045

Fire Particles: https://www.assetstore.unity3d.com/en/#!/content/36825

Water Particles: https://www.assetstore.unity3d.com/en/#!/content/48580

Water sounds: https://www.assetstore.unity3d.com/en/#!/content/14039

Music: https://www.assetstore.unity3d.com/en/#!/content/19233

The bolt from the second practical lecture (although, somewhat adjusted).

Resources from outside the Asset Store:

Hammering sound effect (by cognito perceptu)
https://www.freesound.org/people/cognito%20perceptu/sounds/17012/

Small Explosion (by Ryan Snook)

https://www.freesound.org/people/ryansnook/sounds/110115/

License: https://creativecommons.org/licenses/by-nc/3.0/

Mechanical Printer (by DemonicElectronic)

https://www.freesound.org/people/DemonicElectronic/sounds/166229/