Contents

[Tilemap grid 2](#_Toc127088425)

[Path generator 2](#_Toc127088426)

[Enemy 2](#_Toc127088427)

[Health 2](#_Toc127088428)

[Towers 3](#_Toc127088429)

[Range 3](#_Toc127088430)

[The shooter 3](#_Toc127088431)

[ShopUI 3](#_Toc127088432)

[Building 3](#_Toc127088433)

[Wavemanager 4](#_Toc127088434)

[Inspecting towers 4](#_Toc127088435)

[Upgrades 4](#_Toc127088436)

[Enemy Status effects 4](#_Toc127088437)

[Scriptable objects 4](#_Toc127088438)

[Observer for goldBalance 5](#_Toc127088439)

[And more observers 5](#_Toc127088440)

[Code Review 5](#_Toc127088441)

[UIManager 5](#_Toc127088442)

[SFM pattern 5](#_Toc127088443)

[Strategy pattern 5](#_Toc127088444)

[And even a singleton 5](#_Toc127088445)

Tile map grid  
  
At the start of the project I decided to work on the layout of the map. One of my friends send me a tutorial on how to implement a tile map with conditional tiles. These conditional tiles are a designer friendly way to implement paths and corners within a tile map as they look at their neighbor tiles and change their sprite accordingly. A quick map was created and I went on to make a functional path.

## Path generator

Each conditional tile also had the ability to spawn as a game object. Each conditional tile that is a path now instantiate with the path prefab. This can be seen as a node.

On start the nodes add themselves to a list in the path generator containing all the nodes. Alongside that a simple check happens to see if the node is the first, last or finish node so it doesn’t have to be checked on generation later on.

To make the actual path a ray cast appears in each direction from the path, hitting each aligned node. Then it loops through the list of nodes starting from the first node and gets the aligned notes from the ray cast. It should always have two nodes (unless it’s the last or first) where one of them is the previous checked node and the second one the new one. It loops through all nodes and creates a sorted list where index 0 is the start and index (whatever the number of total nodes is) is the last node.

Path generator is a singleton, there can only be one path in a game. Each enemy’s walker needs a quick and easy one-time reference to get the path created by the path generator. Therefore, it seems like the obvious choice to make the path generator a singleton.

The tile map grid had one problem, it creates itself using start and awake. I’m unable to generate a path on awake and start so this had to be moved to whenever the wave starts.

# Enemy

At this point I was told to think differently, instead of OOP use ECS (Entity Component system). And that’s how I started working on the enemy. The enemy has a walker class which can be inherited from to make more and different walkers. For this project I’ve only made one which is the follow path. (In the future it could also attack towers or make de-routes). The walker only has the path, speed, rotation speed and target node as this should be the case for every walker and is used to determine if they should be targeted by the towers.

The follow path follows the path list as it’s sorted from 0 to max. Each frame it checks the distance between the enemy and the target node and if it’s less than 0,1 it will check if the current target node is either the end or finish (end is end of path, finish is where the damage to the player happens) node and if that isn’t the case it continues to the next node in the list. Creating a sorted list where 0 is the begin of the path.

## Health

I discussed in the lab with my teacher Hans Wichman if it’s ok to use the “health bar” system BTD6 uses by changing the sprite based on how low the health is as I didn’t like seeing health bars on top of the ships and this was approved.

Whenever the enemy takes damage, it tries to load the correct sprite based on its health. This is done through the scriptable object. An enemy can have any amount of sprite and the sprite is based on the difference between the current health and the default health it started with. This can be easily adjusted in the scriptable object it loads from.

# Towers

Once again, I was thinking about ECS. A tower has a Range and shooter(s). Tower itself does not represent that much other than keeping everything together.

## Range

The range of a tower is displayed by a child mesh renderer, it also has properties on its own. The range is a 2d sphere collider and keeps track of which enemies are within range of the tower. They get added to a list when they enter the trigger collider and removed when they are no longer in range. The range also got a method to get a target that a shooter can ask for. There are different targeting modes available but in the final product this has been changed to only have one targeting mode. Even though the code for the others is available.

## The shooter

The tower can have multiple shooters, think about two cannons on one tower. I originally wanted to do this as upgrade instead of increased speed on one single cannon, that’s why shooters are a list. The shooter contains all the data for the projectile. The projectile is just one prefab where the data gets loaded on to after it’s initialized.

One thing to note out is that all projectiles have an AOE, even though one of them doesn’t use this property. It believe this was the better way to do it instead of making a separate class for it. And if you look at BTD6 I think they do it the same way. Some towers have no use for a range in BTD6, even though they still showcase a range indicating that they have one.

# ShopUI

I’ve created a simple shop with a side scrolling UI. It contains itemshopcards prefabs. The itemshopcards prefab can be placed within the shop and it will adjust itself accordingly to make them fit within the shop. The cards only need a reference to the scriptable object of the tower to load the necessary information.

Whenever the user hits the buy button the referenced tower will be sent to the towerplacemanager for building.

# Building

In first few labs the question popped up if we had to use grid or if we were allowed to do free placement like BTD6 and this was approved, as long as the towers couldn’t overlap each other. My first problem was that I had two types of towers, water towers and land towers (and in the future shore towers). The non-conditional tiles can’t have a game object prefab. The way I handled this was by re-making the tile map using different layers. Water, Path, Land. I was already able to collide with path but for the Water and Land tiles I used a tile map collider with each layer having its own tile map collider and corresponsive tag. This way I was able to detect with what tiles I was colliding with.

For placing the towers, I attached a circle collider on the mouse and kept track of the current colliders it’s currently colliding with. To make sure it was only colliding with ground or water I first check if the list equals to 1, if this is the case I try to match the collider’s tag with the tower’s type (land, water) and if this is the case it could be initialized on its location.

# Wave manager

For my intake game I already had a wave manager, this wave manager was awful but the new one originated from it. The wave manager contains a list of waves where each wave exists out of sub waves. A sub wave can have multiple enemies and two variables, when the sub wave starts and how long it will take in total for the wave to spawn. The sub waves are spawned with an enumerator. It will first wait until it’s ready to be spawned, then calculates the time it must wait between the spawning of the enemies. Alongside the spawning it keeps track of the enemies that are alive within the scene. To keep track of when a full wave is finished, I can’t check if the list of enemies is equal to zero as that can also happen when the first enemy of the wave dies. I’m keeping track if all the sub waves are done spawning in a dictionary. Each wave also adds some balance on completion and have a custom “build”/” break” between the wave to make it harder on higher waves.

Whenever a wave is finished there will also be a start wave button appears showcasing what wave the player is on and a timer before it auto-starts. The button can be pressed to skip the timer and instantly start the wave.

There is no generation of waves, this has to be filled in by hand by a designer.

# Inspecting towers

Next up I wanted to work on upgrading towers, to do this you have to inspect/select a tower to see it’s ~~stats and~~ available upgrades. The easiest thing would be to shoot a ray cast based on the mouse input out of the camera on the layer of the tower. Whenever it hits a tower, it showcases the tower on the bottom right using the inspect card prefab. The inspect card manager on the prefab loads in the correct thumbnail, background (based on land type) and the available upgrades within a scroll view.

## Upgrades

The tower upgrade is an abstract class, each upgrade has a cost for each level, this is done with a multiplier per level on top of a base cost. Each upgrade inherits from the abstract tower upgrade class. The upgrades can do anything within their upgrade() method. In the future this could even initiate a second canon. But for now, it’s only statistical improvements.

# Enemy Status effects

One of the towers requirements was to apply a de-buff on the enemy, the status effect should reset if there’s a new bullet. The status effect is a component on top of the enemy. It gets added through the enemy. The enemy first detects if the effect is already active, if this is the case it will extend the effect of the new effect is longer or increase the strength of the effect if it’s stronger. The status effect is an abstract class where new custom effects can inherit from. Now the question came up, when removing the effect how do I keep track of its original value? I could do it on awake/start but what if it’s already being affected by another effect? Then it would reset it to a permanent de-buff. After some discussion I concluded to use scriptable objects for that.

# Scriptable objects

I’m using scriptable objects to get the basic data of a tower and enemy. The scriptable objects are stored as a reference in the enemy and tower prefabs. This created two new methods where I can do LoadData() on the enemy and the tower. They will both copy the values from the scriptable objects.

Enemy is just a prefab and thanks to the scriptable object it’s easy to make new enemies. I’ve made like 8 different enemies.

# Observer for gold Balance

My shop has no references at all, it’s purely based on observers. It listens to whenever an enemy dies to add a new balance and subtracts whenever there is a purchase. It also invokes an event when money is changed. This is because the buy buttons will be updated all around the projects. It calculates if the button can be pressed or not. Another event invokes when the balance gets added. This one exists so I can showcase how much a enemy drops on death or the end of a wave’s completion bonus gives.

## And more observers

I’m using a few more observers:  
When the end of a wave is reached.  
When all the waves are completed, to showcase a win condition.  
When a enemy reaches the finish, and needs to damage the gate.  
When a enemy dies, to remove it from the enemies alive list in the wave manager.

# Code Review

I thought I already implemented 3 patterns, the singleton, observers and a strategy pattern. According to Wiebe I did not have the strategy pattern as it requires an interface. This caused me to try and implement it in the project but there wasn’t a good fit for it. If I really wanted to get a third pattern, I had to implement something new. Wiebe also said that I wasn’t using my game manger’s singleton to its max potential, so I decided to change my wave manager to a singleton instead of a reference via the game manager. Other than that, my review was quite positive.

# UIManager

I already had a UI manager called Mouse Manager which changed from state based on what the user was doing (Shopping, Building etc.). This was originally just a Enum to keep track of what UI should be open. I wanted to implement a mechanic that will auto close the UI pop-ups to make the playfield clear. For example, clicking next to the shop’s UI will close it. After re-following the lectures, I quickly realized I could use a SFM pattern for this.

## SFM pattern

The SFM patterns has an exit and enter method where the state opens and closes its own UI. Not only that, but it also changes the behavior of what the mouse is looking for so it can detect when it didn’t click on UI to close it. This allows me to make tower inspecting great as the user can now click next to a tower to close the “inspecting” state’s UI.

## Strategy pattern

During the implementation I also saw an opportunity to implement the strategy pattern! Each state needs to have an exit and enter method; therefore, I could easily use the strategy pattern by turning the basic state in to an interface.

## And even a singleton

Now originally, I used the game manager to get a reference to the mouse manager, but I decided to rename the mouse manager to UI manager as it felt like a better fit. Also, it didn’t make much sense to me to keep it via game manager, so I decided to make it a singleton to easily change the state throughout the project.