Pixel Tower Defense starting guide

This guide contains all needed info for you to get started with the package. We will set up a custom level with waves of enemies, a path and tower pads. At the end of this guide you will be able to create your own levels, enemies and towers.

Also notice that this project has a public Trello board. You can follow the project there.

Contents (Ctrl+Click to jump):

Overview

When you import the package, you'll see the following folder structure:

Fonts – all the fonts that are used throughout the game.

Graphics – graphic files for all the prefabs that are used in the game. Inner folder structure is self-explanatory.

Prefabs – all the prefabs.

Auto setup – pre- "built" prefabs that are automatically added to the scene using the "Setup level" feature (more on this later).

Enemies – prefabs for monsters, including animators, animator overrides and common animations.

Levels – prefabs to create your own levels. Including background grid and ornaments (trees, houses etc.).

Tower Pad Popup – common popup menu stuff. Including choice packs and choice items (more on this later).

Towers – tower prefabs, including bullets and animations

UI – User Interface related prefabs

Utils – utility prefabs (behavior only)

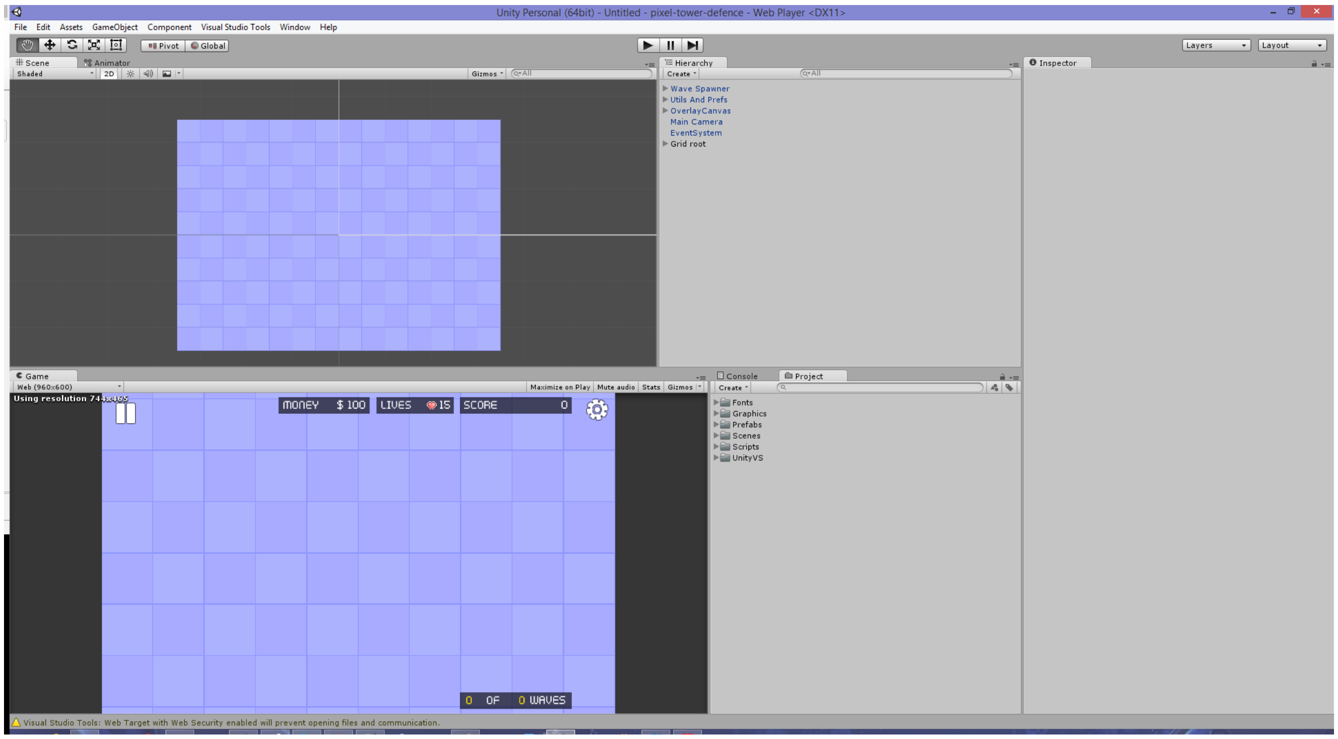
Scenes – all levels, menu and level selection scene.

Scripts – all the scripts that make the game happen.

Setting up your level

To create a new level, follow these steps:

1. Create a new Unity Scene (Ctrl + N or File → New Scene)
2. Through the menu, go GameObject → Pixel Tower Defense → Setup level. The window will open.
3. In the window, chose desired level style, for example **Badlands**
4. Press **Setup**.
5. You'll see a set of prefabs that were added to the scene. Let's look at them and then create a path. You can also save your scene now.



Visual Studio Tools: Web Target with Web Security enabled will prevent opening files and communication.

Fig. 1. Initial scene setup

You'll see the following scene structure:
Wave spawner – This is where you will tweak the wave spawning process.
Utils and prefs – helper prefabs that are needed to be on the scene. Contains tower choice prefs, properties holder (HashIDs), Setup Level info and Tower pad provider. More about each one later.
OverlayCanvas – uGUI canvas that contains needed screen windows and UI items.
Main Camera and **EventSystem** – usual Unity stuff
Grid root – parent object for the background grid.

If you start your scene now, an error will occur and playing will stop: “*Seems that there's no starting point. Please run the tile linking script..*” that's because we haven't created our path and a base tower yet. Let's do that now. In the

Project tab open *Prefabs* →*Towers* folder. You'll find the Base prefab there (see Fig.2). Drag it somewhere in the scene. Your base is now properly placed (see Fig.3). We should now add some path tiles so enemies will be able to walk. All tiles are located in a theme-specific folder. For example, badlands path tiles are located in *Prefabs*→*Levels*→*Badlands*. You'll see 7 different prefabs, 6 of which are tiles. You should now choose which tiles to drag. It's suggested that all tiles will be parented to some empty “Path” game object in the scene (Fig.4.).

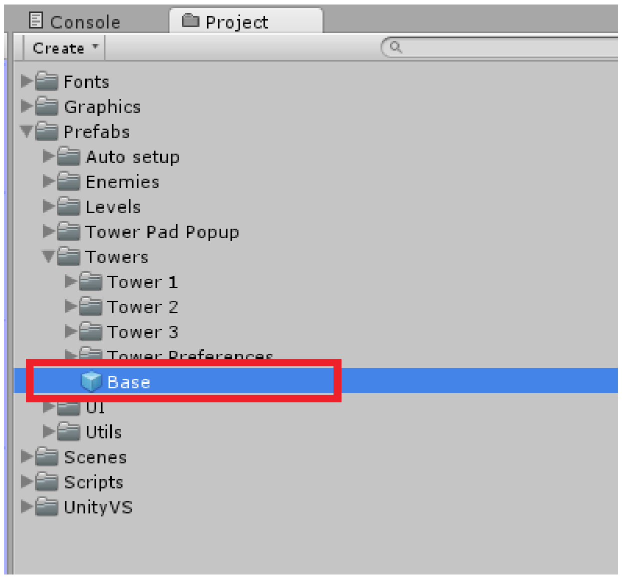


Fig.2. Base tower prefab location.

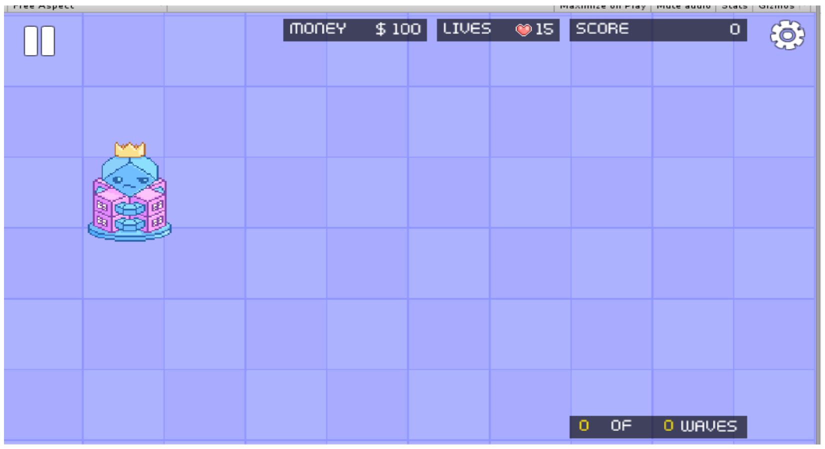


Fig.3. Base tower in the game view.

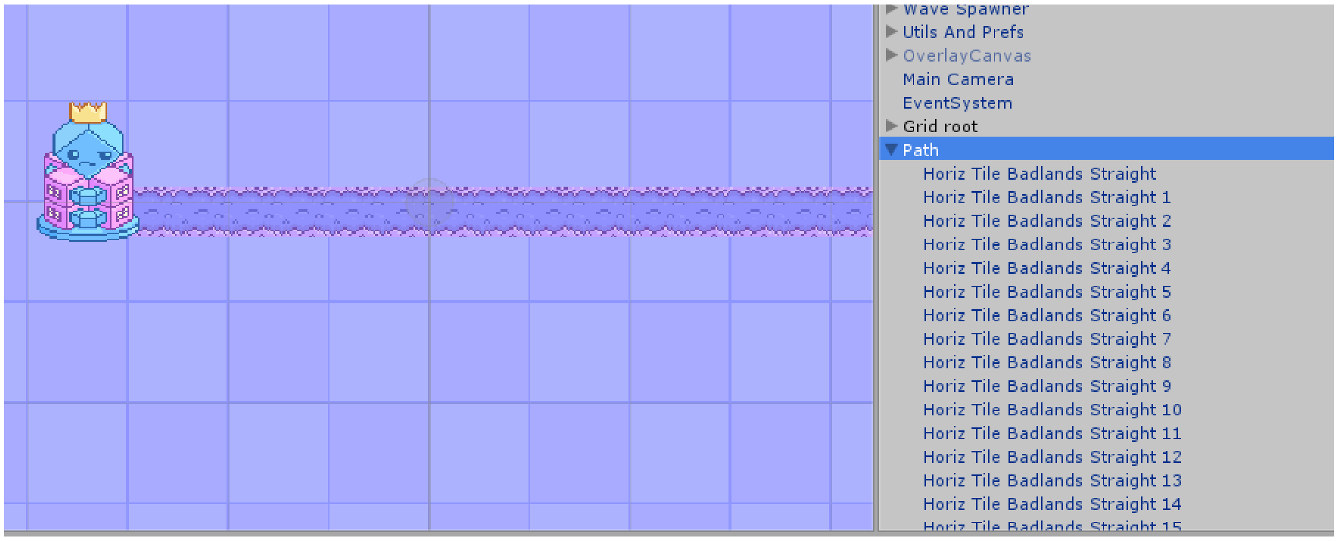
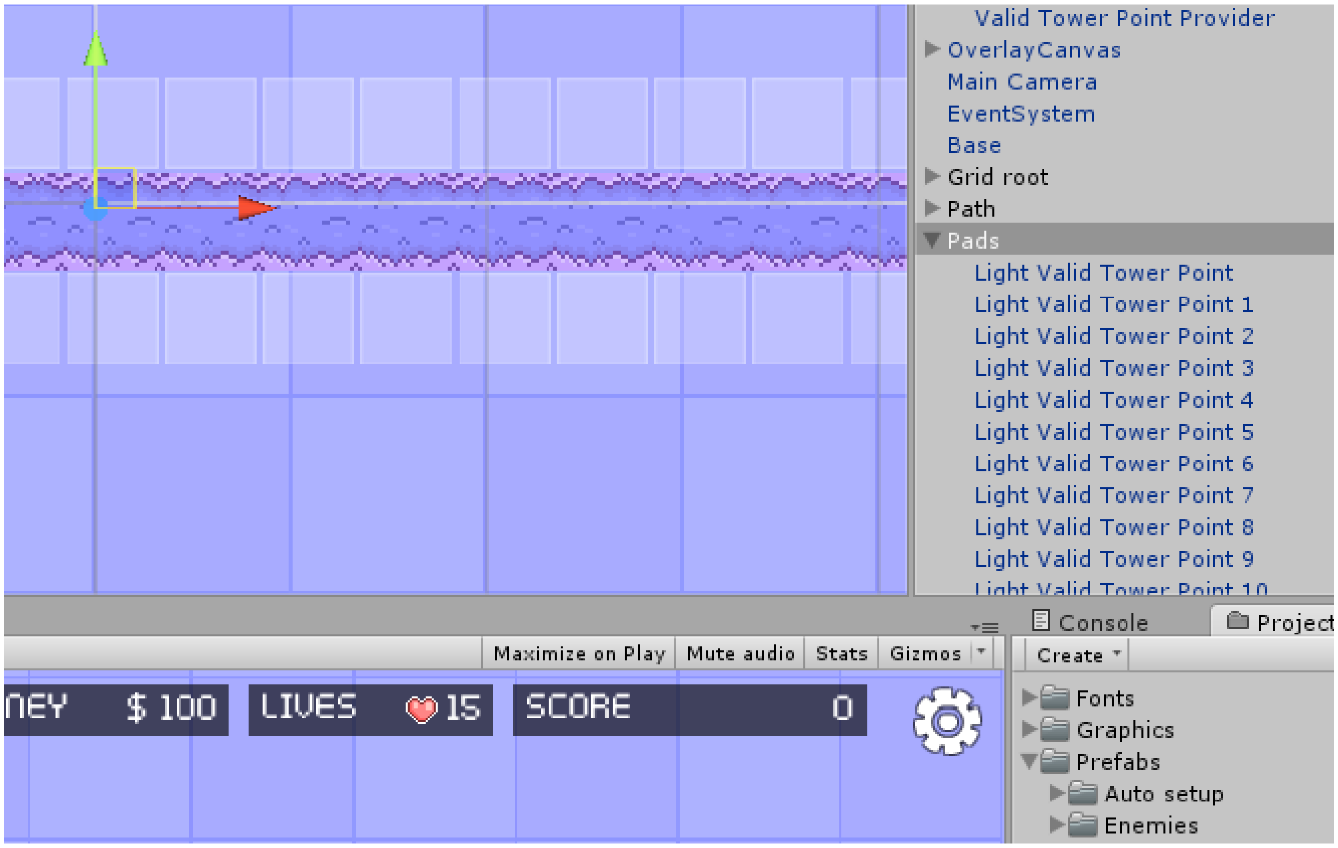

Wave Spawner
Utils And Prefs
OverlayCanvas
Main Camera
EventSystem
Grid root
Path
Horiz Tile Badlands Straight
Horiz Tile Badlands Straight 1
Horiz Tile Badlands Straight 2
Horiz Tile Badlands Straight 3
Horiz Tile Badlands Straight 4
Horiz Tile Badlands Straight 5
Horiz Tile Badlands Straight 6
Horiz Tile Badlands Straight 7
Horiz Tile Badlands Straight 8
Horiz Tile Badlands Straight 9
Horiz Tile Badlands Straight 10
Horiz Tile Badlands Straight 11
Horiz Tile Badlands Straight 12
Horiz Tile Badlands Straight 13
Horiz Tile Badlands Straight 14
Horiz Tile Badlands Straight 15

Fig.4. Path

You can also make something more complicated than just a straight line. But for the sake of readme it will be enough. **One last thing** to do before you can start your game is to run the Path Tile Linking script via the *Game Object*→*Pixel Tower*

Defense→Link path tiles. Run it now. If everything is done correctly, you'll see a message in the console: “*Path tiles were linked*”. You can now safely run the game and see how the base tower gets destroyed (we don't have any defense towers yet).

It's time to add some tower pads, which indicate places where towers can be placed. You'll find two different pads in the *Prefabs→Levels.* As we are using Badlands theme, we'll take the *Light Valid Tower Point.* Again, it's recommended to parent these pads to some empty game object. Add some tiles to the scene. **Note:** *you can make them snap using the Ctrl button. They will be spaced properly if you'll move them while pressing the Ctrl button.* See Fig. 5.



Valid Tower Point Provider
OverlayCanvas
Main Camera
EventSystem
Base
Grid root
Path
Pads
Light Valid Tower Point
Light Valid Tower Point 1
Light Valid Tower Point 2
Light Valid Tower Point 3
Light Valid Tower Point 4
Light Valid Tower Point 5
Light Valid Tower Point 6
Light Valid Tower Point 7
Light Valid Tower Point 8
Light Valid Tower Point 9
Light Valid Tower Point 10
Console
Project
Maximize on Play
Mute audio
Stats
Gizmos
Create
NEY $ 100
LIVES 15
SCORE 0
Fonts
Graphics
Prefabs
Auto setup
Enemies

Fig.5. Tower pads.

One last (but **very important**) thing to do is to link the *Light Valid Tower Point* prefab to the *Valid Tower Point Provider,* which is located inside the empty *Utils and Prefs* game object (scene hierarchy) (Fig.6). All other themes use dark valid tower points, so there will be no need of overriding the prefab. You may ask why is there such a difficulty. It's actually the fastest and safest way to provide that prefab. There are infinite number of solutions and this is just one of them.

If you start the game now, you will be able you to buy and sell towers, which means the level is now fully playable. In the next section we'll see different settings that we can tweak.

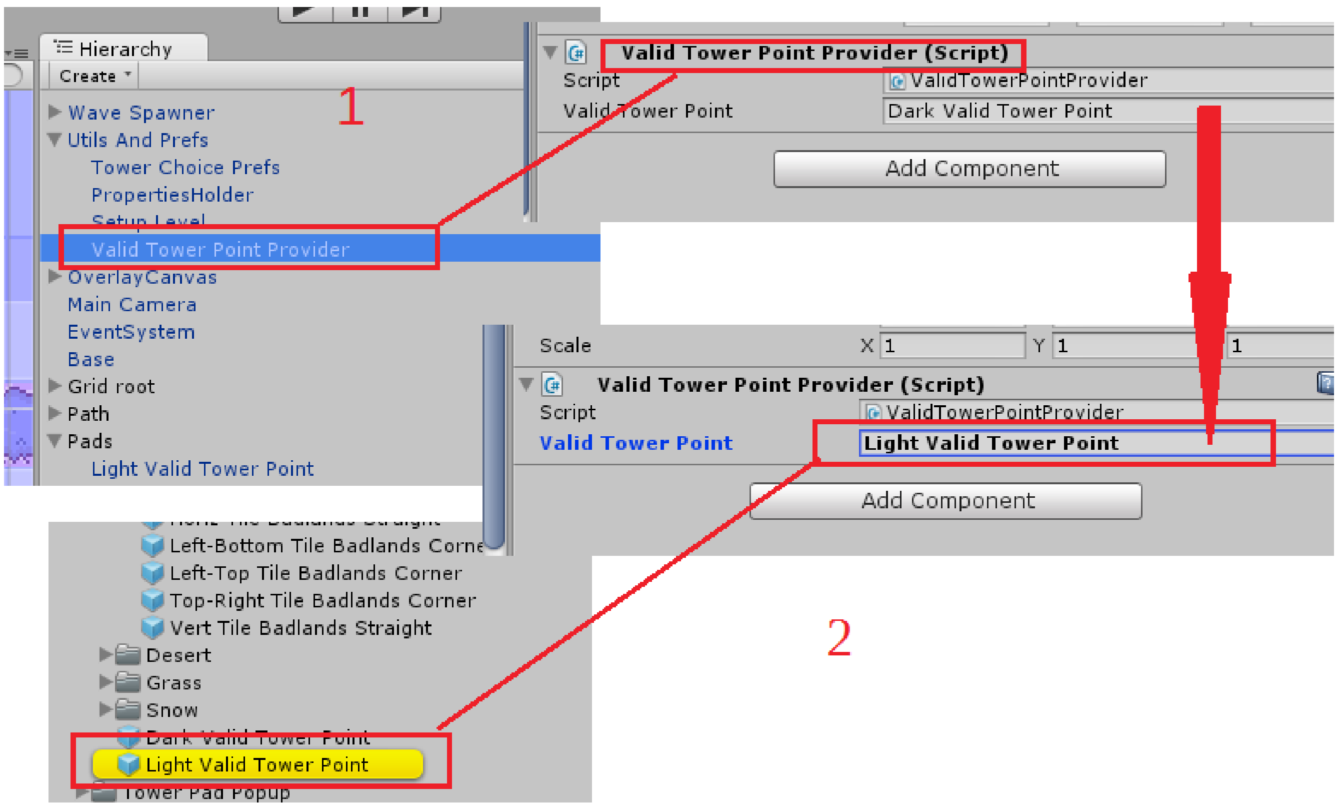


Fig.6. Changing the Valid Tower Point Provider.

Different balance settings

Waves and appearances

There is a number of setting that you can tweak to make the game hard or easy. The settings include:

- Tower rank preferences. How each rank of each tower is different from another. This includes **Damage**, **Fire rate** and **Range**.
- Enemy preferences. The **Speed** and **Health** of each monster.
- Waves preferences. **Number** of enemies and their **Spawn rate**. This also includes the number of **Waves** and different Enemies (**Appearances**) from that Wave.

This is how it works. The level has **Waves**, each Wave has **Appearances**, each Appearance has spawning **Enemies**. So each level can have several waves, each wave can have different appearances. Appearance is a description of how many enemies of which type should be spawned and at what rate.

IMPORTANT NOTE: Each appearance should be used by ONLY ONE Wave. If you want to have two identical appearances – make two copies!

Let's look at Waves first. Take a look at the Wave Spawner game object in the scene hierarchy (Fig. 7).

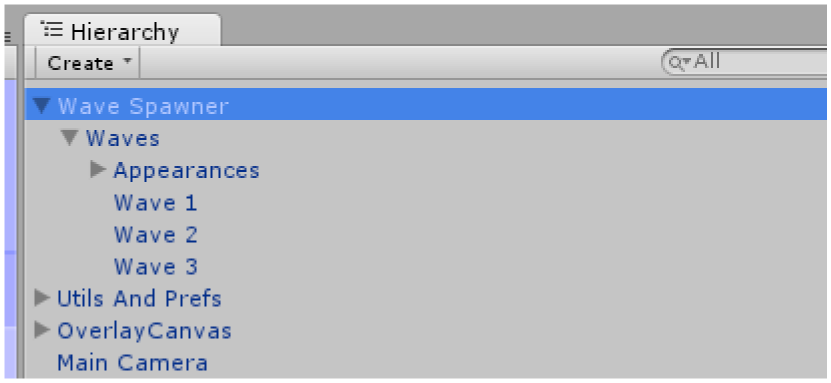
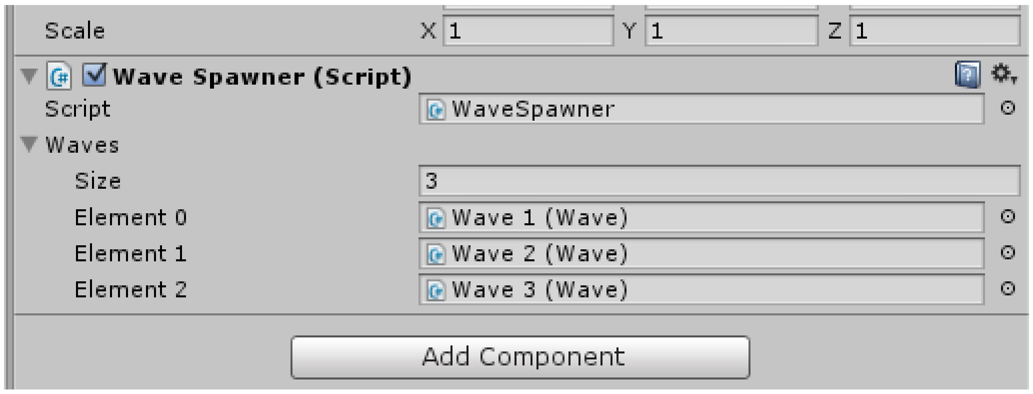


Fig.7. Wave spawner in the hierarchy.

It contains an array of linked waves (Fig. 8). The waves themselves are parented to the Wave Spawner object (Fig.7).



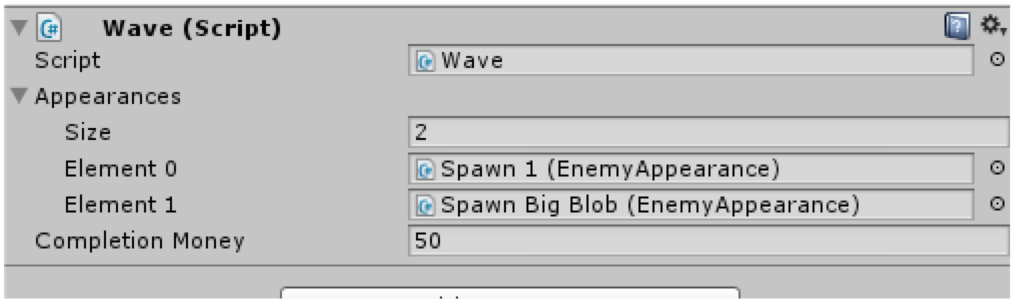
Scale	X	1	Y	1	Z	1

Wave Spawner (Script)

Script	WaveSpawner
Waves	
Size	3
Element 0	Wave 1 (Wave)
Element 1	Wave 2 (Wave)
Element 2	Wave 3 (Wave)

Fig.8. Wave spawner properties.

Now select any wave and look at it's properties.



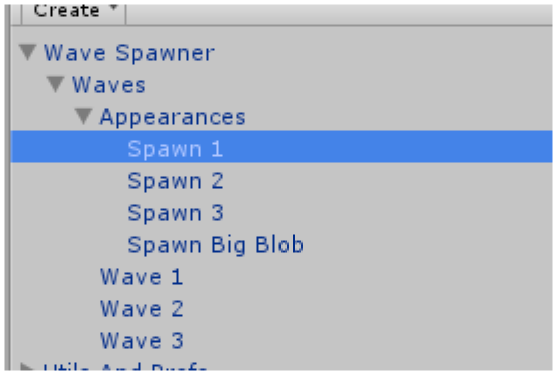
Wave (Script)

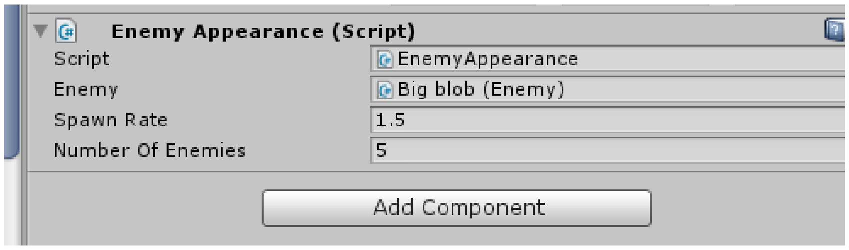
Script	Wave
Appearances	
Size	2
Element 0	Spawn 1 (EnemyAppearance)
Element 1	Spawn Big Blob (EnemyAppearance)
Completion Money	50

Fig.9. Wave properties.

It has a set of **Appearances** – a set of spawning characteristics of ONE type of enemy. Waves can have different types of enemies in them, and in that case there will be several appearances. It also has a **Competition Money** property which specifies how much money the player will get once every enemy in the Wave is dead.

Now let's look at Appearances (Fig. 10 and Fig. 11).





Enemy Appearance (Script)

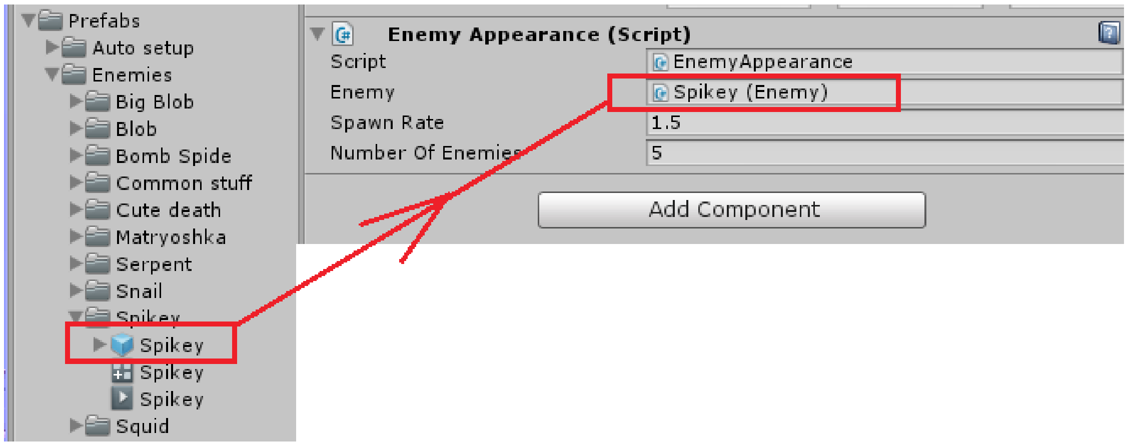
Script	EnemyAppearance
Enemy	Big blob (Enemy)
Spawn Rate	1.5
Number Of Enemies	5

Fig.10. Appearances in Hierarchy

Fig.11. Appearance properties

Each appearance has a link to the **Enemy** prefab, **Spawn Rate** (in seconds) and total **Number Of Enemies**. Tweaking these values, adding different appearances to the waves and adding waves to the wave spawner will enable you to create your own uniquely balanced level.

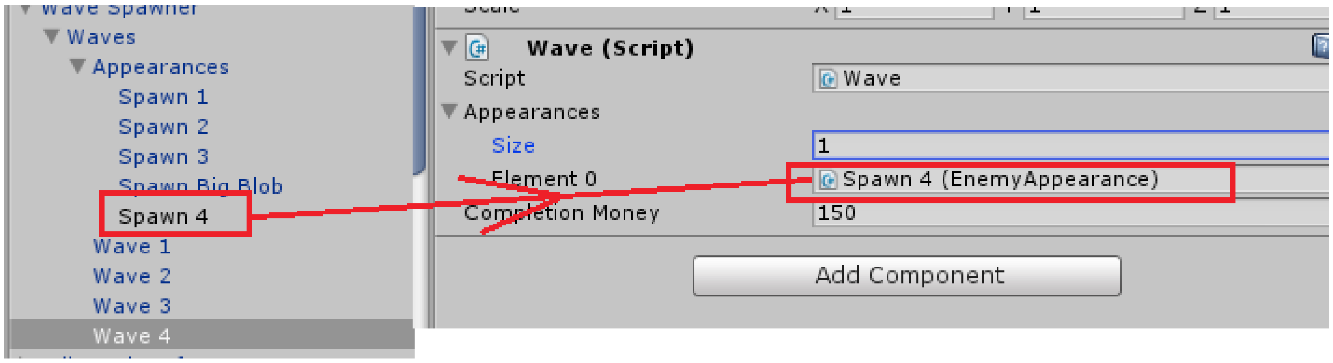
Let's create another appearance. The easiest way is to duplicate any of the "*Spawn..*" objects in the Scene using *Ctrl+D*. You'll have something like "Spawn 4". Now, in the Project view in the *Prefabs→Enemies* folder open any monster folder and drag the prefab to the **Enemy** field in the appearance (Fig. 12).



Script EnemyAppearance
Enemy Spikey (Enemy)
Spawn Rate 1.5
Number Of Enemies 5

Fig. 12. Adding a different enemy type to the appearance.

You can also change the **Spawn Rate** and **Number Of Enemies**. Now add a new **Wave** by duplicating any of the existing ones and drag your newly created appearance to the Appearances array. If the number of appearances is more than 1, simply change the **Size** property to 1 (Fig. 13)



Script Wave
Appearances
Size 1
Element 0 Spawn 4 (EnemyAppearance)
Completion Money 150

Fig.13. New wave.

One last thing to do is to add our new **Wave** to the **Wave Spawner**. Select the Wave Spawner and change the **Size** property to 4. Then drag our newly created **Wave 4** to the last field (Fig. 14). That's it, you're done. You've just added a new Wave to your level.

Please note: *in future releases this process of cross-dragging will be improved to be more user oriented, with "Add / Remove" buttons and more nice-looking windows. Stay tuned for later updates*

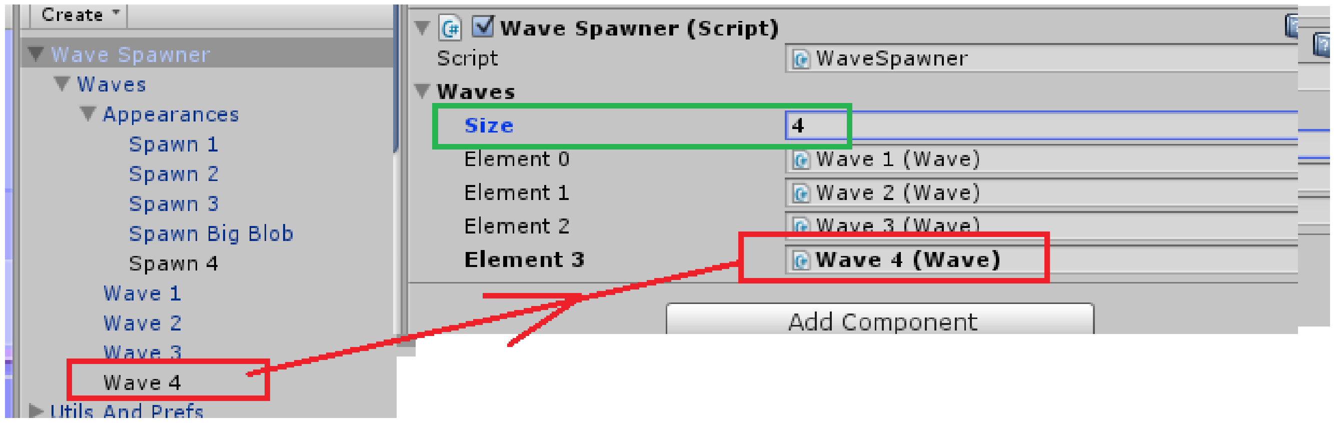
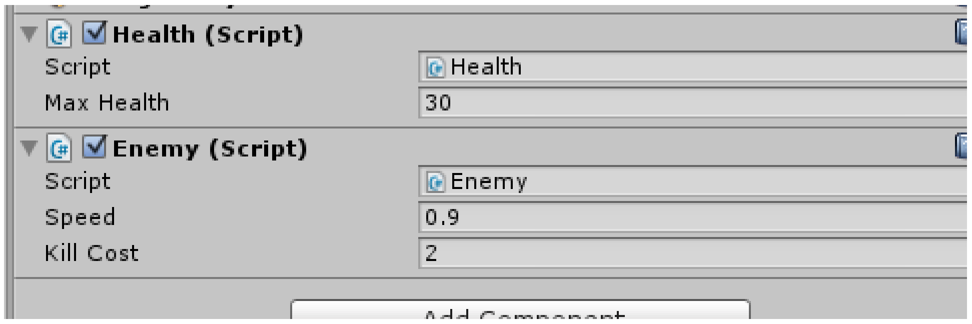


Fig. 14. Adding a new Wave to the Level.

Enemy properties

Let's now look at some properties that enemies have (Fig. 15).



Health (Script)

Script	Health
Max Health	30

Enemy (Script)

Script	Enemy
Speed	0.9
Kill Cost	2

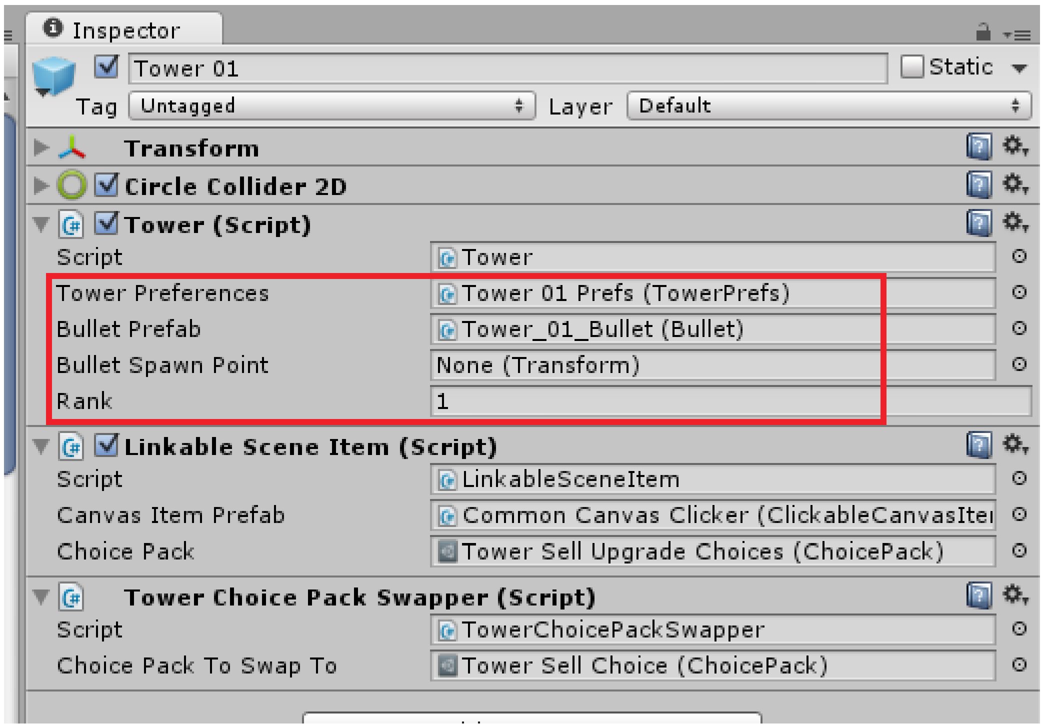
Fig.15. Enemy properties.

The **Health** script contains logic for current health value and that's where the "toughness" of the enemy is tweaked. **Max Health** property is self explanatory. In the Enemy script there are two properties – **Speed** and **Kill Cost**. The **Speed** specifies how fast the enemy is (it's measured in Units per Second). The **Kill Cost** specifies how much money the player will get for killing this type of enemy.

Tower properties

Tower properties are a bit more complicated. Each tower has several ranks, which have different properties. So instead of wiring the properties to the Tower, we wire them to **Tower Prefs** which have four (or an arbitrary number of) **Rank Prefs**, and only then we connect the Tower with the Tower Prefs. At runtime, the tower asks Tower Prefs for the prefs of the needed rank. This way any tower can have any preferences at any rank.

Let's look at the Tower itself now (Fig. 16). Right now, we only need the Tower script (we'll look at **Linkable Scene Item** and a **Tower Choice Pack Swapper** a bit later).


Inspector

Tower 01 Static

Tag Untagged Layer Default

Transform

Circle Collider 2D

Tower (Script)

Script	Tower
Tower Preferences	Tower 01 Prefs (TowerPrefs)
Bullet Prefab	Tower_01_Bullet (Bullet)
Bullet Spawn Point	None (Transform)
Rank	1

Linkable Scene Item (Script)

Script	LinkableSceneItem
Canvas Item Prefab	Common Canvas Clicker (ClickableCanvasIte
Choice Pack	Tower Sell Upgrade Choices (ChoicePack)

Tower Choice Pack Swapper (Script)

Script	TowerChoicePackSwapper
Choice Pack To Swap To	Tower Sell Choice (ChoicePack)

Fig. 16. Tower properties.

The definition of each property:

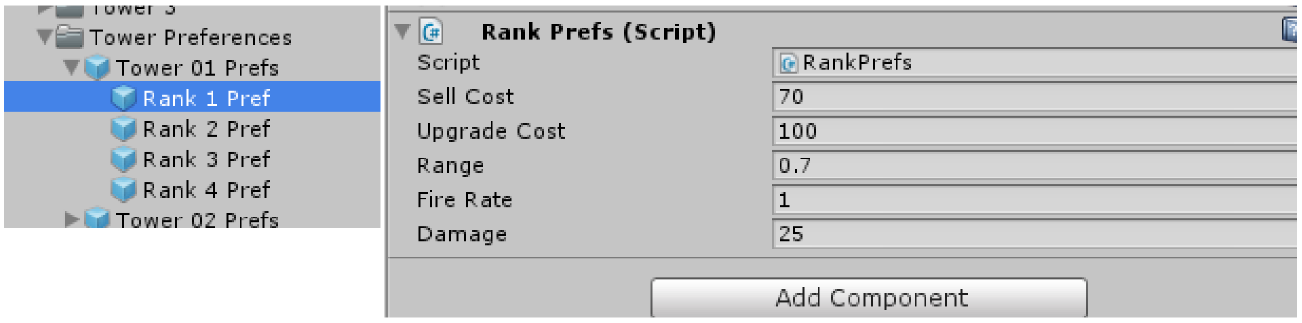
Tower Preferences – a link to the prefs object, which contains all info about prefs of different ranks of the tower

Bullet Prefab – a link to the bullet prefab which will be spawned when firing.

Bullet Spawn Point – a transform point, relative to the Tower itself, on which the bullet will be instantiated. It can be null, and in that case, the bullet will fire from the center of the tower.

Rank – current (and starting) rank of the **Tower**.

Let's now look at the tower preferences. Click on the object inside the field or go to Prefabs→Towers→Tower Preferences. If you click on **Tower 01 Prefs** for example, you'll see that it only contains an array of **Rank prefs**. Unfold the prefab and see the children of the Tower Prefs object. Select one of them, say, **Rank 1** and look at it's properties (Fig. 17).



Script	RankPrefs
Sell Cost	70
Upgrade Cost	100
Range	0.7
Fire Rate	1
Damage	25

Fig. 17. Rank pereferences.

You'll find the following properties there:

Sell Cost – the cost of selling the tower on this rank

Upgrade Cost – the cost of upgrading the tower to the next rank

PLEASE NOTE: The last rank should have the upgrade cost of -1, which specifies that that is the last rank and we won't be able to upgrade the tower any further.

Range – active radius of the tower (it's often so the range increases with each rank)

Fire Rate – time between shots, in seconds. The lower – the better (1 means one second, 2 means two seconds)

Damage – self explanatory.

How it all works

Now I will tell you some secrets of how it all works (the code is fully commented, but it's good to have a big picture in mind).

The most simple things are enemy / towers logic.

Enemies follow the path, because each tile knows the next one. This allows enemies to recalculate the direction only once in a while. This is what happens when you run the Path linking script – each tile get's to know it's neighbor.

Towers use simple collider logic to know when there's an enemy inside of their shooting range. They will track all the enemies inside their range so if an enemy dies or goes outside of the shooting range, the towers pick up another one without sphere casting or other costly physics things.

When enemies collide with the Base Tower, they call the destruction logic, which decreases the number of lives of the Player.

The most interesting and important stuff is the choice menu clicking logic. Every clickable scene item should respond to clicks (or touches) somehow. One of possible solutions is to have colliders on scene items and then use raycasts on each touch. This project uses a slightly different approach – it utilizes the uGUI interaction system. Each scene item that can be clicked has an invisible *canvas double* (see *ClickableCanvasItem* class) (graphically it can be something completely different). And that *double* uses a standard uGUI Event system to respond to touches and clicks. These clicks are transferred back to the scene item, allowing a scene item to process them. The towers and tower pads use exactly this logic. As you've probably noticed, both tower pads and towers respond to clicks almost the same – they trigger the opening of the choice menu. Each clickable scene item has a *Choice Pack* that consists of *Choice Items*. When clicked, the scene item triggers the opening of the round menu. This menu shows each item from the choice pack and waits for any of them to be clicked. When it happens, it launches that choice item logic on current tile, that triggered the opening menu itself. For example – an item that buys a tower destroys the tower pad and places the selected tower on its place. The tower upgrader doesn't destroy anything – it simply increases the tower rank if the player has enough money. You can check existing Choice Packs and

Choice Items inside the Prefabs →Tower Pad Popup→Choice Packs folder. Most of items in that folder are self-explanatory. You can also check out the code for more info (there's a lot of comments there).

Useful public links

Feel free to drop me a line at the cyrill@nadezhdin.org
Or visit my blog http://blog.nadezhdin.org/

You can also follow this project's public Tello board
https://trello.com/b/YFdFCxAH/pixel-tower-defence