Turn/tile based framework manual

Contact: simplevideogamesdev@gmail.com

I. Intro

The core of this framework is a turn based system where any number of players can take turns performing actions with their units(s), and a tile based system where these units can be placed or moved on the map. No matter if you want to make a roguelike, strategy, tactics or a tabletop game, as long as it has player(s) taking turns, and a map made out of tiles, this is the framework you were looking for!
The framework contains a set of tools that you might need to develop your game, and will also streamline your process by giving you components to adjust, instead of building everything from the ground up. It contains map, turn and ui managers all directed by the game manager gameobject, as well as classes like player, unit, resource, tile etc. and will extend your editor so you can add new instances of these classes in the same way you would add a cube or a camera to your scene. Most of these are optional (if your game doesn't require resources for example, you can just leave that part out) and there is a demo scene to get you started with a playable demo as soon as you download the framework.

II. Importing the package into your project

Download and import the package from the asset store. All of the assets are packed into the Turn and Tile based framework folder. The assets are sorted by type into the folders. You could directly change any of these files (any change to the files in the Resources folder, will change the default classes generated by the editor extensions but more on that later), but it's probably best if you don't change anything in the frameworks folder. Any new assets that you develop should be kept in the root assets folder or sorted into new folders there.

1. Using the demo scene

You can load the demo scene (found in the Scenes folder) and test it right away, before importing the framework into your own scene, or use the demo scene as a starting point to build your game.

Yes the scene looks empty with only the GameManager game object in it, but when you run the game, it will generate a map and 2 player game objects each with their starting unit somewhere on the map(a different colored pawn). It might also populate the map with any number of resource spawners (yellow and black checkers) that spawn resources on nearby tiles after a few turns. You can pick these resources up with any of your units and that resource will be awarded to the owner of the unit. The GameManager might also populate the map with any number of "neutral" hostile units (or neutral buildings that spawn these units), that will wonder the map until they attack a player unit. A UIManager game object will also be created, it will display a canvas with some basic information about the current turn, player and selected unit as well as allow you to do some basic input. The end turn passes the turn to the next player, but be careful if the timer runs out (top right slider represents the timer) your turn will be automatically ended. Starting unit of the player whose turn it is will be selected, represented by the purple circle underneath it, and you can move to the one of the tiles highlighted by a green circle (safe tile) or a red circle (tile where someone can attack you). If you are in range of an enemy unit, it will be highlighted by a yellow circle, and you can attack it. Another thing that you can do is click on the build and then house buttons at the bottom of the screen when you have one of your pawns selected then click on a adjacent tile to build a house there. You can later use the house to train more pawns and build up your army. Don't forget that you need to have enough resources to build or train a new unit, and that you can do that only once per turn. The game ends when one of the players loses all of his units, and is defeated.

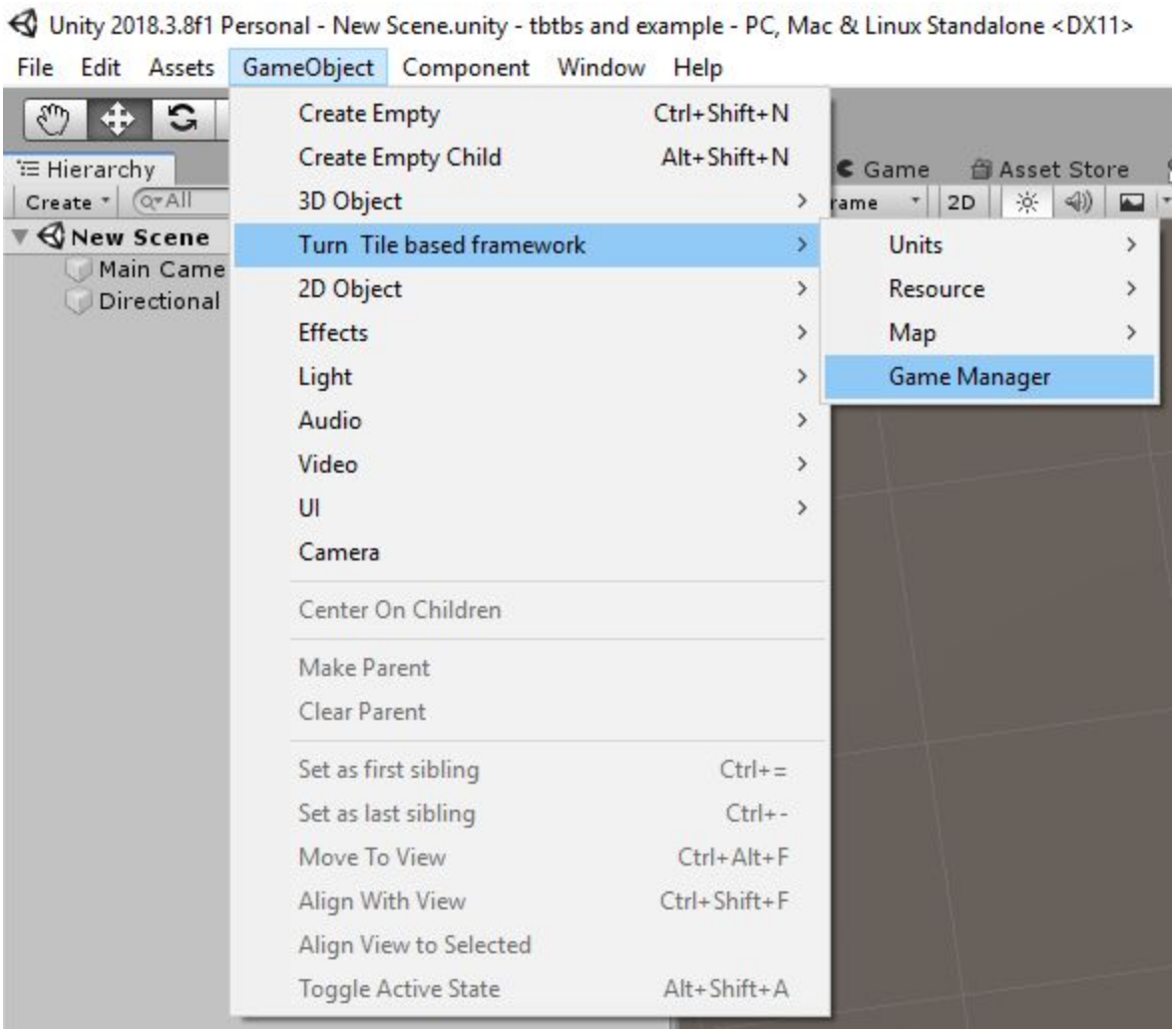
In the next chapters we will talk about customizing content and mechanics, but let's take a moment to talk about how the GameManager works. It instantiates TurnManager that keeps track of time and turns, and UIManager that contains ui settings and the main canvas with all the info and buttons. It also instantiates a map gameobject, that has all the tiles, neutral units, and player starting points parented to it, as well as a player container gameobject that has all the player game objects and their units parented to it. We will look into how each of these function later, for now just remember that it's important to keep this gameobject hierarchy consistent when building upon this project.

2. Importing the framework into your scene

You can use this framework in any environment that you want. You still need to use the tile prefab for the walkable terrain (it's explained how you can customize it in the map chapter), but you can have any game objects in the background as well as your own lighting and skyboxes. If you have a scene that you want to start with, follow these steps, but if you are going to start with an empty project, you might want to start off with the demo scene and build from there.

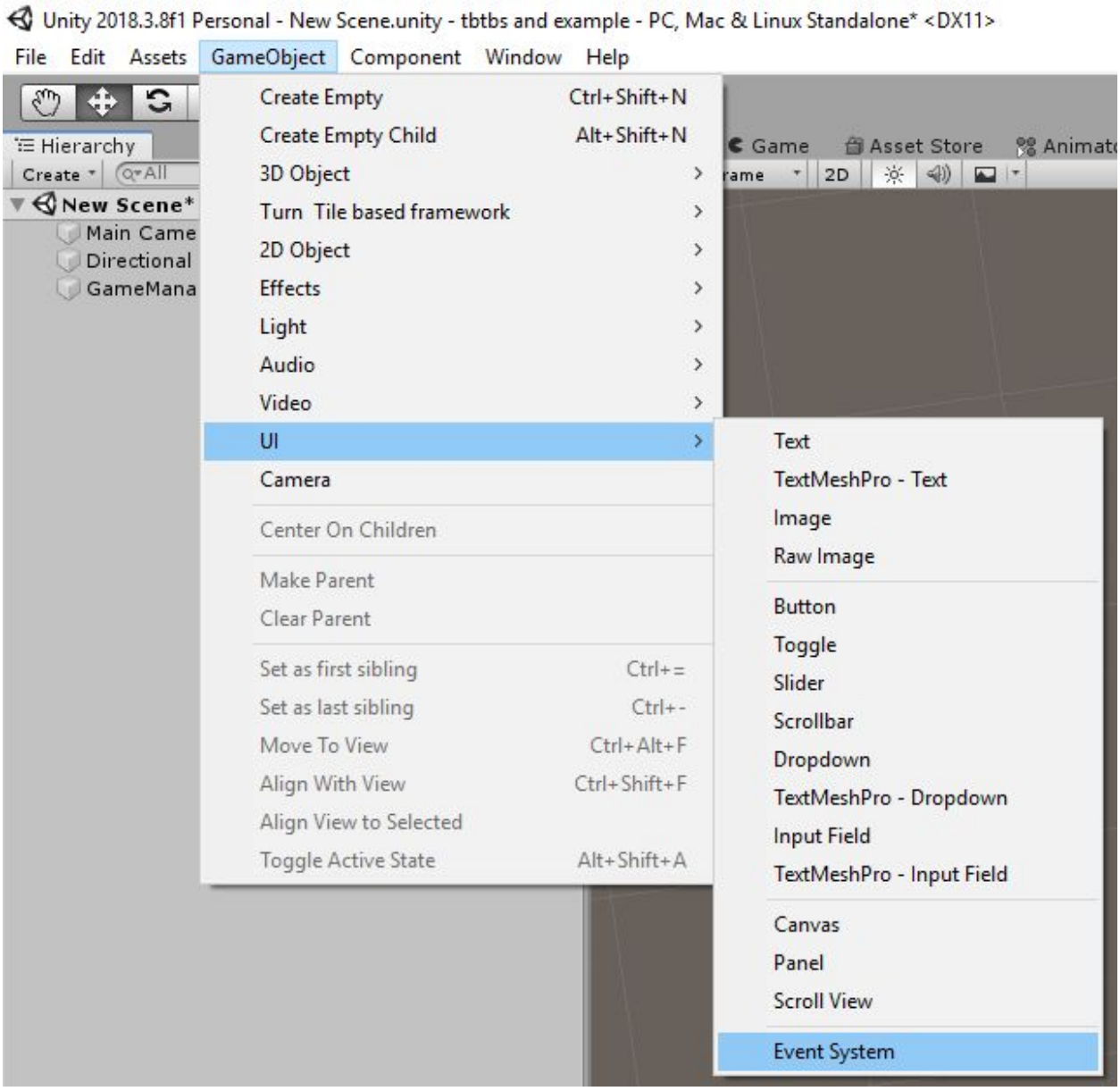
A. GameManager

Press the add gamemanager button (GameObject>>Turn Tile based framework>>GameManager)



B. EventSystem

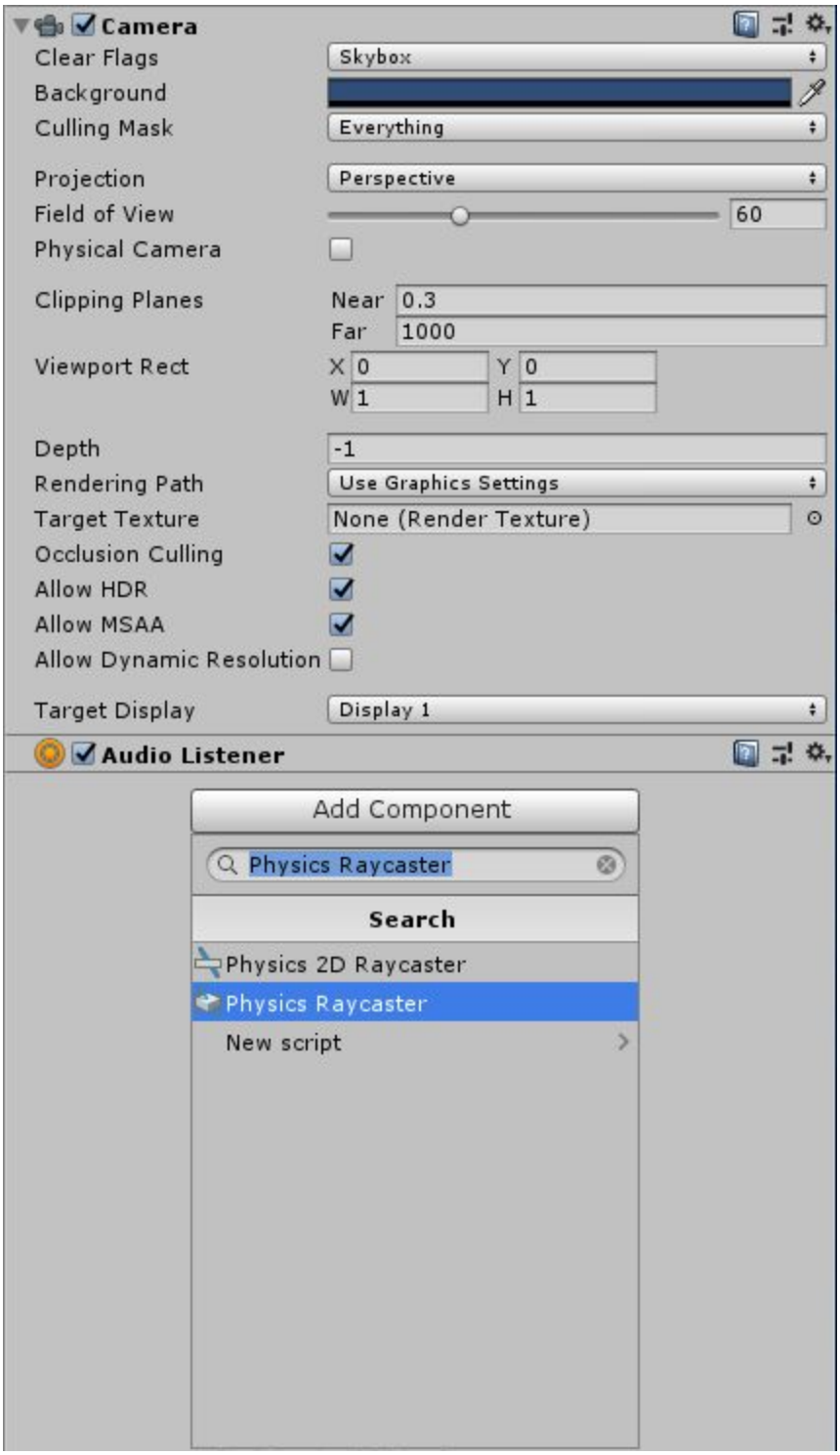
Press the add event system button



C. Setting up the camera

You can use any type of camera with this asset, a static camera (just remember to point it at the map), a rts style camera or even a 1st or 3rd person camera with a bit of adapting. That is why I didn't include a camera script in this framework (I personally use a free asset called "RTS Camera" found at https://assetstore.unity.com/packages/tools/camera/rts-camera-43321. You can download and import it in minutes, then just add the script called rts camera to the camera

in the scene). The important thing is that you need to add a Physical Raycaster component to the camera (you can find the Add component button at the bottom of the inspector and search for the component).



Now if you run the scene the game manager will generate a map for 2 players with starting units, that you can control using the UI provided. Your lighting, the skybox and any camera effects will still be applied. If you want to use your own models for units, resources and tiles, or use a premade map with other models in the background read on.

III. Building your own units

Now the unit setup is the bread and butter of the framework. The demo scene has a troop unit, that can build a building unit, that can train more troop units. It also populates the map with some neutral units. When you are using a randomly generated map you can uncheck the UseRandomlyGeneratedNeutrals checkbox found in the GameManager inspector, but if you are using a premade map, just don't place any on the map. Neutral units are hostile to all players and will try to destroy them. These are just examples, but you can use them as prototypes to build upon your own player and neutral units.

1. Types of units

A. Player troops

Player controlled units that appear in the training menu.

B. Player buildings

Player controlled units that appear in the building menu.

C. Neutral troops

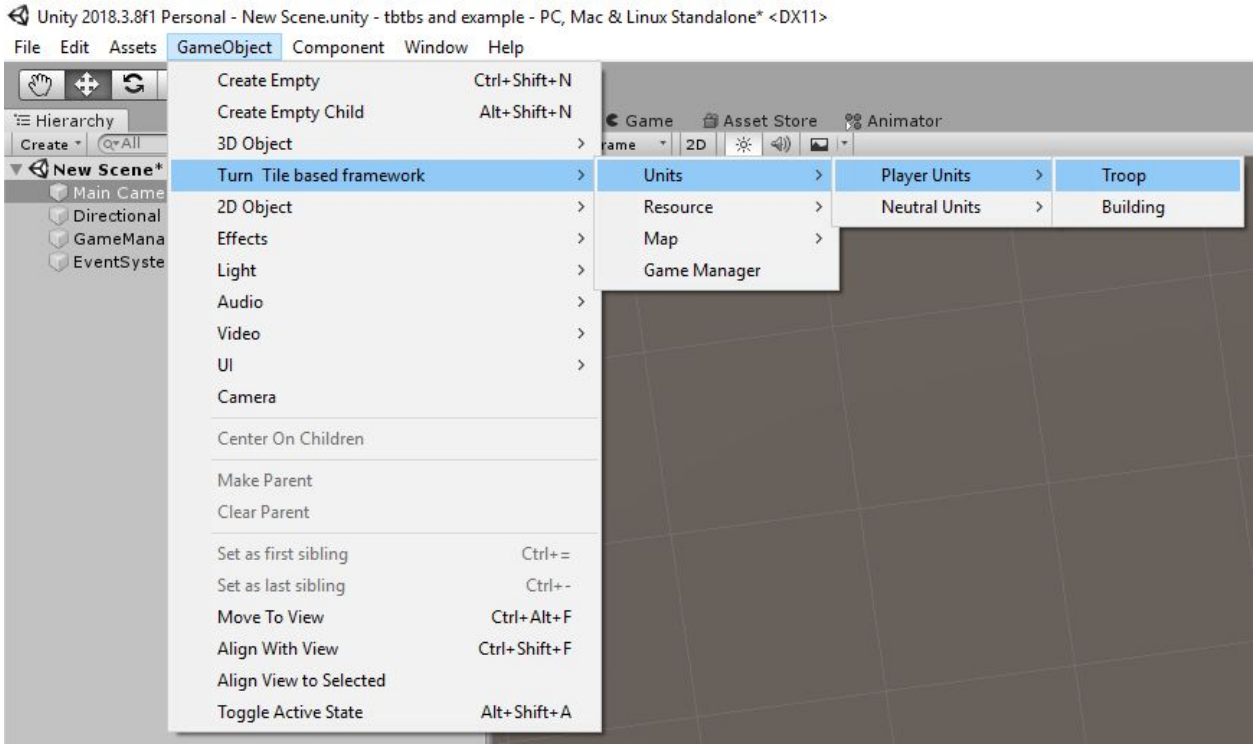
AI controlled units that are trained in the neutral buildings.

D. Neutral buildings

AI controlled units that train troops when they have enough resources.

2. Creating units

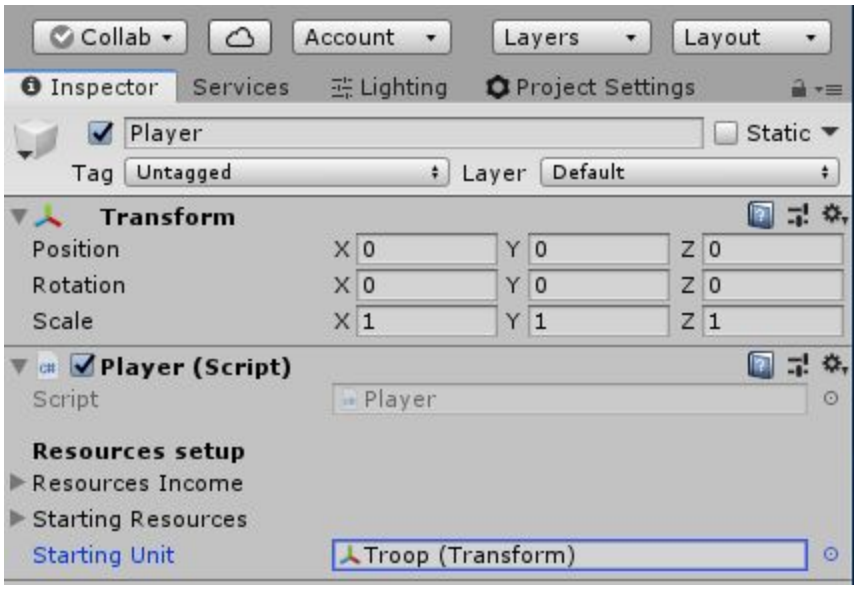
To create a new unit use the units menu in the GameObject menu item as shown below (GameObject>>Turn Tile based framework>>Units).



This will create a default unit of the chosen type and add it to your scene. To add this unit to the game you want to make a prefab out of it, so just rename the gameobject, make a folder for all of the unit types that you will create and drag it into the folder creating a prefab. You can delete the original copy that is in your scene, and repeat this process for any other unit types that you want to create. The new unit has default behaviour and appearance and we will go into changing that a bit later.

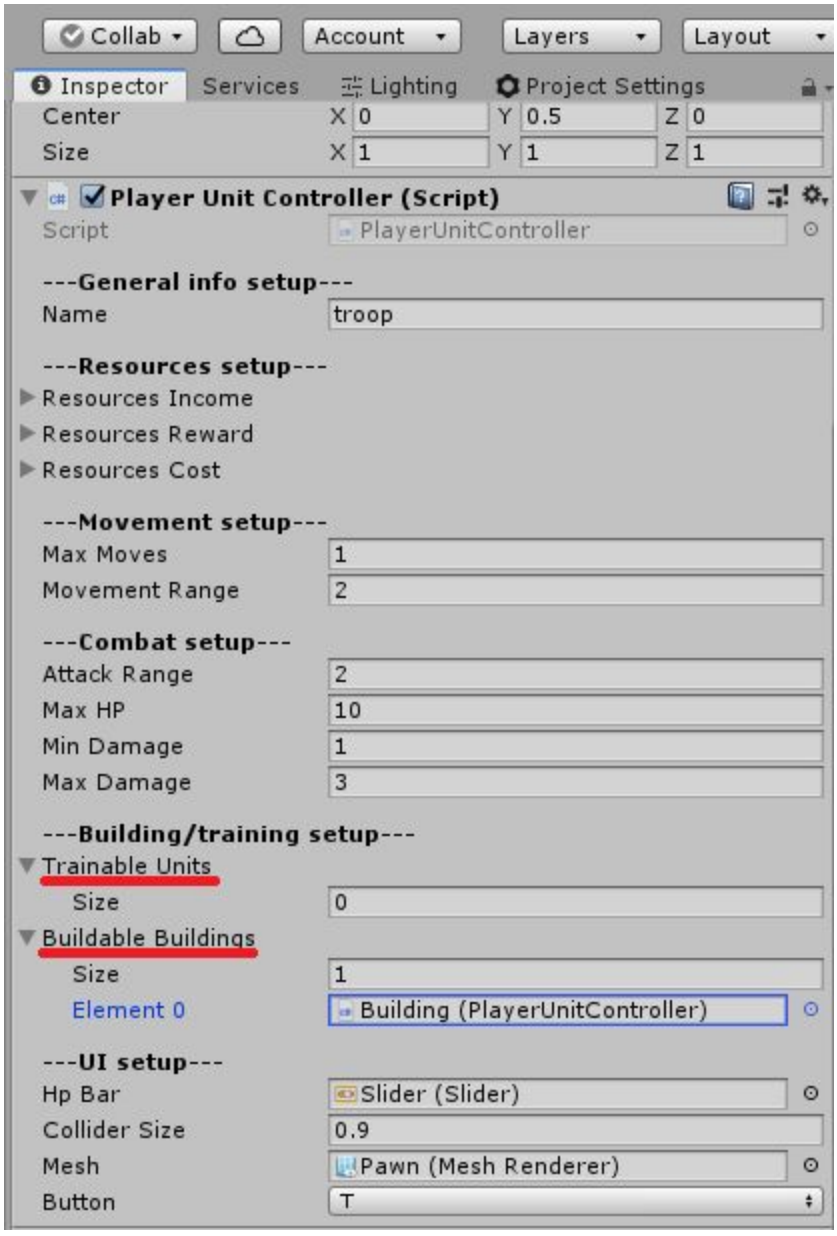
3. Using units

You can use the unit prefab you just created as a player starting unit. To do so just drag the prefab you made into the startingUnit variable in GameManager inspector.



Another thing you can do with the prefab is allow another unit to train or build it. So pick a unit (for example the startingPlayerUnit) and open it up for editing. You can find a list of trainable and buildable units in the inspector window of the PlayerUnitController (or the NeutralUnitController script, but more on that later). If you add the new unit prefab you made

earlier to one of those lists, and run the game, the button to place that unit will appear ingame.



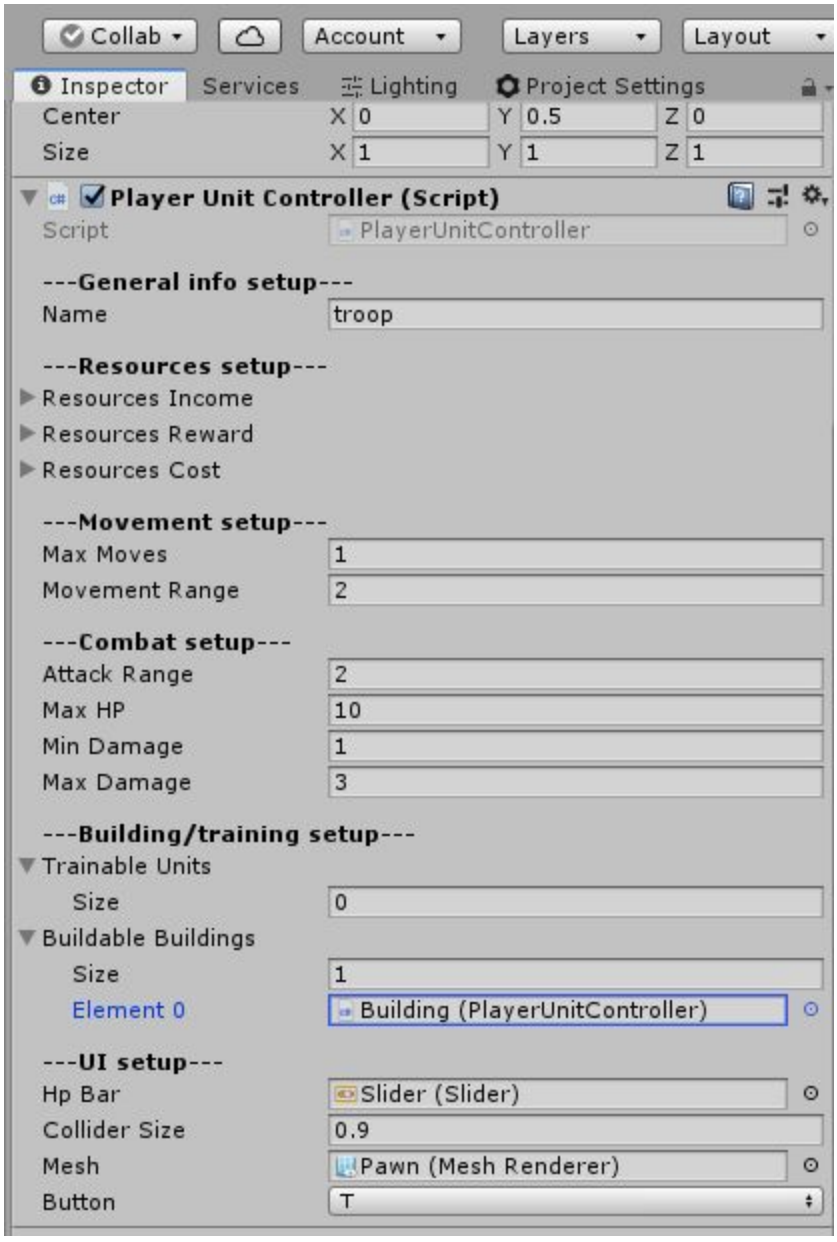
If the player has enough resources, click the button, then click on an empty tile next to the selected unit, and the unit will be instantiated at that position. You can't move or attack with it this turn, the player has to wait for his next turn to use the unit.

4. Customizing units

Unit behaviour is controlled by 3 different scripts. The UnitController script has properties and behaviours that apply to all unit types, but it actually isn't attached to any gameobject in the scene. The PlayerUnitController script extends the UnitController script, and it inherits

everything from it. On top of that it has behaviours that let the player control its actions during his turn. It is attached to any player controlled unit, and you can consult the inspector window of unit prefab for both UnitController and PlayerUnitController options. The same applies for the NeutralUnitController script , except on top of inheriting UnitController behaviours it contains behaviours that let the unit decide and execute actions after each player finishes his turn.

A. Unit behaviour



Name: unique unit identifier(must be lower-case)

Resource: There are several lists of resources to setup. Each resource is composed of a name and an amount.

-Cost: This represents the amount of each resource that needs to be paid to build or train this unit.

-Income: This represents the amount of each resource that the unit generates for its player owner each turn

-Reward:This represents the amount of each resource that the player who destroys this unit is rewarded.

Movement: You can change the number of moves that a unit can make in a single turn, and change the range that a unit can move in one of those moves.

Combat: You can change the health and attack range of the unit and the minimum and maximum damage the unit deals per attack.

Building and training: Unit prefabs in these lists can be trained or build by this unit.

UI: UI setup part of the unit script contains a reference to the units health bar, the size of the clickable collider, the hotkey to build or train the unit and a reference to the units mesh (this is used for coloring the unit to the player owner color).

B. Unit appearance

All of the unit prefabs come with a model parented to the main gameobject, but you can replace this model, along with its textures with anything you like, as long as you parent it to the main gameobject. The main object has a box collider, that needs to be the same size as the TilePrefab collider, so scale down your model so that it fits into that box. The script changes the first material in the materials array of the MeshRenderer component, to the material that has the same color as the player color that you can set up in the GameManager game object inspector. You just need to drag the models gameobject that has a MeshRenderer attached to it into the Mesh field in that units inspector. If you want to skip this step just leave the field empty, and also do the same for neutral units. Animations aren't directly supported as of yet, but you can implement an animation controller for any model, and add a idle animation without coding. For attack, building and training animations you need to add a trigger to the Attack or TrainOrBuild functions found in the Unit.cs script, so that the animation controller knows when to play these animations. Movement animations would be more tricky. You would need to implement navigation agents, and bake the navigation areas before changing the Move function found in the Unit.cs script. Easier animation implementation and movement animations are at the top of the list of things to be added to the framework, but if you still need help implementing them or want to contribute to the framework, you can find my contact at the top of this document.

IV. Setting up the resources

A Resource is a simple class that contains its name and quantity. Every player has a list of resources they can spend on more units and every neutral unit has one as well. The framework also contains classes for pickupable resources, and resource spawners that come with a model attached to them and are placed on the map by the map generator. If you don't want it to do so just uncheck the UseRandomlyGeneratedResources checkbox found in the GameManager inspector.

1. Starting

Starting resources are awarded to each player and neutral unit at the start of the game. You set this up in the player prefabs, player controller script inspector. So for example if you wanted players to start with 100 gold and 25 wood, you would make a list of 2 elements in the starting resources list, naming the first one "Gold" and setting its value to 100, and naming the 2nd element "Wood" and setting its value to 25. Every resource that the player is able to earn, pickup or get awarded should be listed in the starting resource list, even if he starts with none of

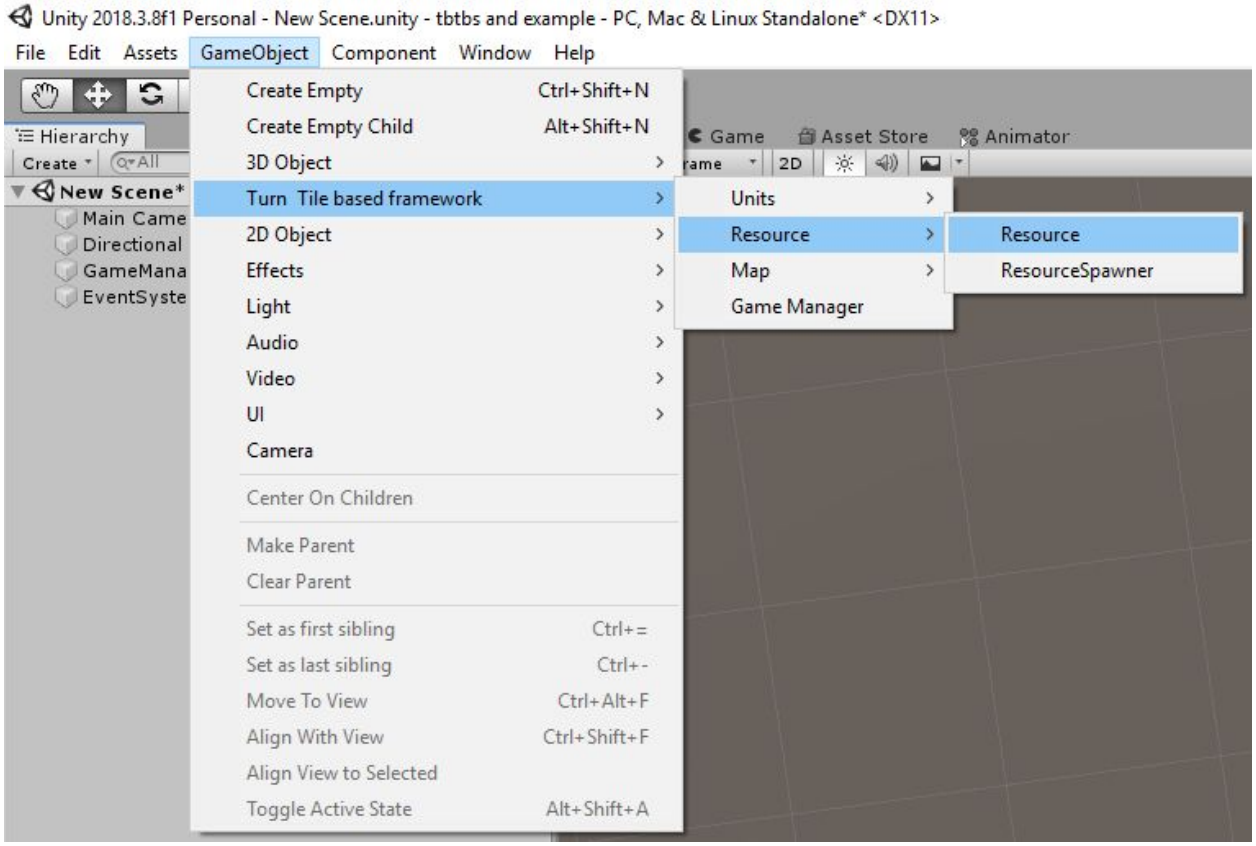
those resources. So if you want to add stone, but don't want the players to start with any, just add "Stone" to the list, and set it's value to zero.

2. Gain

Resource gain represents how much of each resource do all players gain passively each turn. This setting is found in the player script inspector window attached to the player prefab.

3. Pickupable

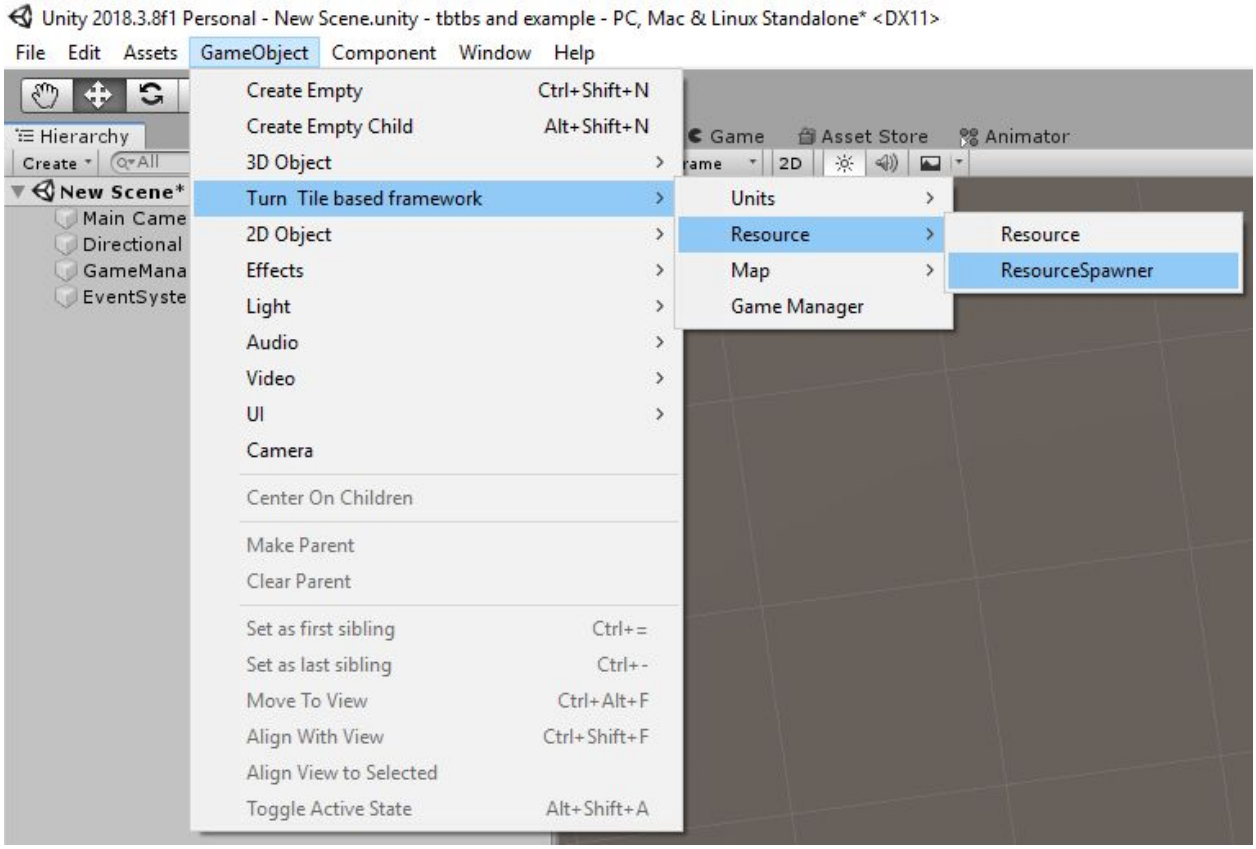
A pickupable resource in this framework is a prefabed gameobject with a ResourceController script attached to it. It is placed on the map by either the map generator or a resource spawner, and can be picked up by any unit that moves onto the same tile as the pickupable resource. When it is picked up, it will reward the unit owner with the resources from a list located in the ResourceController script.
To create a new pickupable resource use the resource menu item tab (GameObject>>Turn Tile based framework>>Resource>>Resource).



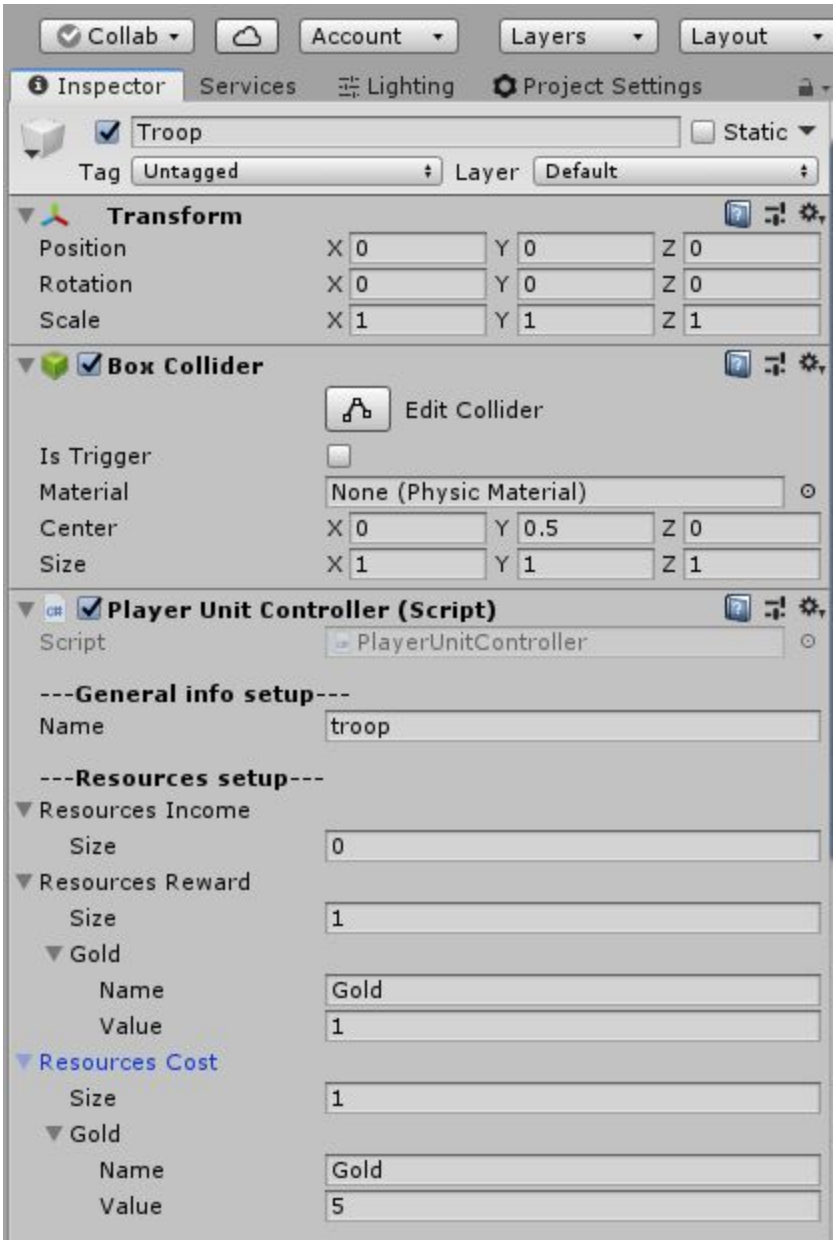
This will add a new pickupable resource to your scene and you should prefab it in the same way you did for unit creation. To use it ingame either add it to the MapManager prefab list called “Resources Prefab”, or place it somewhere in the premade map prefab (more on this in the map chapter). You can customize the model and material used by the prefab in the same way as for the unit prefab (just open the prefab in prefab editor mode, and keep the new model parented to the gameobject with the ResourceController script, scaling it down to fit inside the box collider).

4. Spawner

Resource spawners are prefabs that have a chance to instantiate a pickupable resource on tiles close to it. You can change the Spawn Range in the ResourceSpawner script, as well as the time it takes to spawn a new resource on the map. Just have the resource prefab you want to be spawned in the Resource Prefab field of the script.
To create a new resource spawner use the resource menu item tab (GameObject>>Turn Tile based framework>>Resource>>ResourceSpawner). You prefab, edit and use in the same way you do for the pickupable resource.



5. Cost income and rewards



You can find these options in any unit prefabs inspector window. **Cost** is paid when you train the unit, **income** is awarded every turn per unit to the player, and **reward** is awarded to the player that destroys the unit.

6. Neutrals

Neutral units don't have a owner that stores and spends resources, but they can do that for themself. Each neutral unit has a list of resources that can be increased and spent on training more units. A neutral unit will always train or build any unit it can afford instead of moving.

After you setup income, reward and cost resources for a neutral unit, you also have to set the Starting Resources list. These are the maximum resources awarded to the neutral unit at the start of the game in a similar way as player starting resources (The actual amount of resources is anywhere from zero to the amount you put into the starting resources list. This is done so that the players can't know exactly when the neutral units will train a new unit).

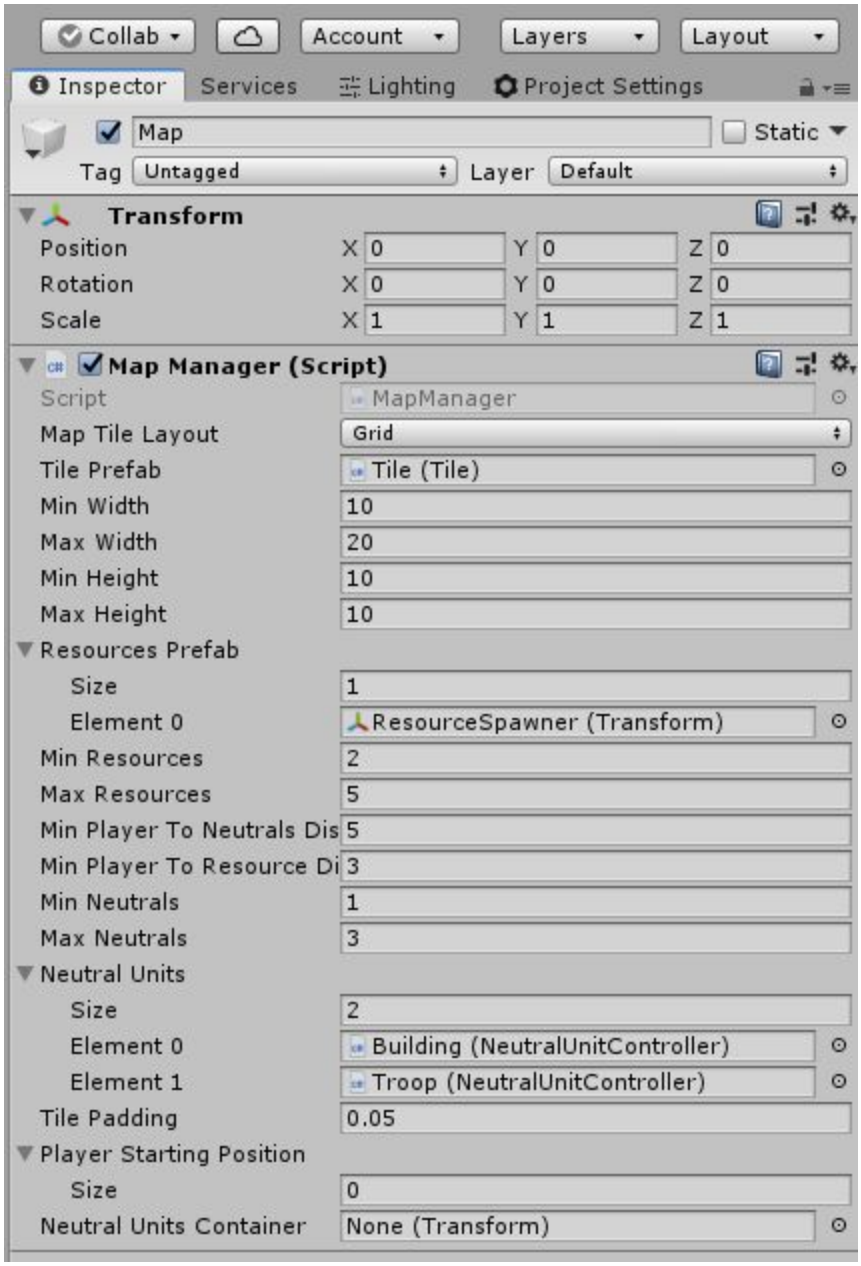
V. Setting up the map

This framework works with a premade or a randomly generated map. You control this by simply ticking a checkbox called useRandomlyGeneratedMap in Game Manager on or off. If it is on, the Game Manager will generate a matrix of tile prefabs that units can walk on, but if you leave it off, it will instantiate the map prefab as a whole.

1. Randomly generated map

The random map generator will do some of the work for you, and will help your games replayability. It will generate tiles, player starting positions, neutral units, resources and resources spawners based on the options that you give it. The Game Manager has a whole part of its inspector dedicated to generated map options like the size of the map, the padding between tiles, number of resources and neutral units spawned and distance to players for them, as well as prefabs of those resources and units. It also keeps a reference to the tile prefab that you can edit in prefab edit mode. Player starting positions list and neutral units container are used only for the premade maps, so you can ignore them for now. This framework supports grid and hex tile layouts. If you want to use hex tile layouts you also need to change tile prefab to one that is hex shaped (you can find an example one at Assets/Turn and Tile based

framework/Resources/Map/HexTile).



2. Premade map

Another option that you have is to make the map yourself. To create a new map use the map editor menu button (GameObject>>Turn Tile based framework>>Map>>PremadeMapGrid or PremadeMapHex depending on the layout type that you want). When you prefab it and open it up for editing, you will be presented with the default map. You can duplicate any tile, and move them around as long as they are connected and don't overlap. If you are making a grid map

layout, keep everything snapped to 1 unit on the x and z axes. If you are making a hex map layout keep the snapping on the x axis: 0,86602 and on the z axis: 0,5. You need as many player starting positions as the maximum number of players on this map, and you need to add them to the Player Starting Position list found in the Map Manager script inspector on the map prefab gameobject. They are basically empty game objects (you can duplicate the existing ones). You can also add resources and resource spawners on top of any tile, and the same goes for neutral units (but they need to be parented to the NeutralUnitsContainer gameobject that is provided with the default map)

3. Tiles

You can create many different types of tiles to use both for the premade and the randomly generated maps. You can create a tile using the editor menu button (GameObject>>Turn Tile based framework>>Map>>Tile or HexTile), prefab and edit it.

VI. Setting up the ui

The UIManager is a game object generated by the GameManager when you run the game and has the UIManager script attached to it. It also contains the canvas with all the panels and buttons.

1. Highlighting system

Highlighting system marks the selected unit, units that the selected unit can attack, and tiles where the selected unit can move as well as if those tiles are safe or dangerous. This is done with 4 different prefabs that you can fully customize. You can find these prefabs in Assets>>Turn and Tile based framework>>Prefabs>>UI>>HighlightSystem.

2. Ingame display

Ingame display is located at the top left corner of the screen. It displays information about the player whose turn it is currently. To edit this panel’s appearance you can open Assets>>Turn and Tile based framework>>Prefabs>>UI>>UIManager and look for the panel named Ingame display

3. Action bar

Action bar is located at the bottom of the screen. It displays information about the currently selected unit, and lets you perform various actions with it. The action button lets you move to a tile or attack an enemy unit (but you can use right click to avoid using this button). The build and train buttons lead you to a new menu with all the units that the selected unit can build or train and displaying some information about these units. You can edit this panel's appearance in the same way as the ingame display, but the action bar has a few different panels parented to it: MainMode, TrainMode and BuildMode. To edit these panels you first need to make them active so that they are visible, and when you are done disable all but the MainMode, later the script will activate and deactivate these as needed. All of these panels have a Horizontal layout group component attached to them, that allows them to control the layout of the buttons generated depending on the units available to build or train by this unit, so don't remove this component. If you want to edit the buttons themself you can find the prefab in Assets>>Turn and Tile based framework>>Prefabs>>UI>>Button

4. Turn control

The last part of the ui are the EndTurn button and the Timer slider. You should be able to edit these with ease.

VII. Customizing the framework for your project needs

1. Player

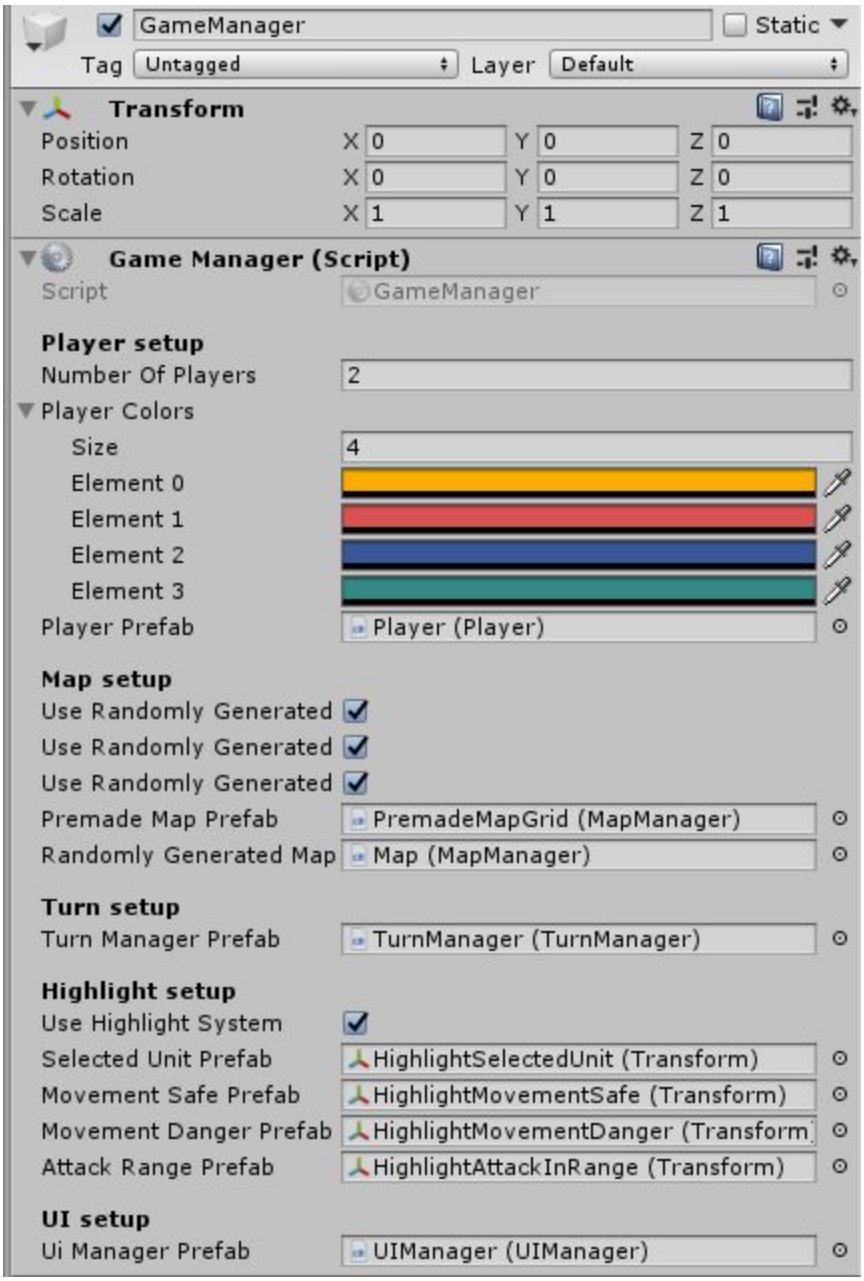
The Player prefab is a empty game object with the Player script attached to it and has a UnitManager gameobject parented to it. The UnitManager contains all the units that belong to that player, and all future units that the player trains or builds will be parented to it. The Player script is where you set up starting resources and resources income, as well as the starting unit for the player.

2. TurnManager

TurnManager is just a game object with the TurnManager script attached to it. It keeps track of the turn sequence, and times the turns. The only thing you can adjust here is the timer that ends your turn when it runs out.

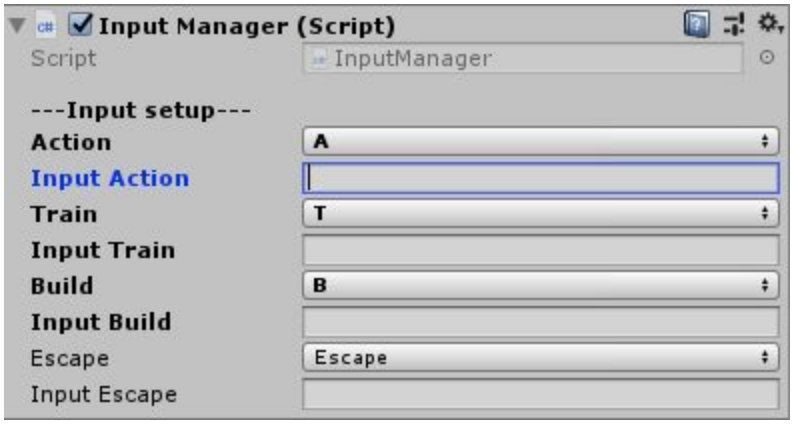
3. GameManager

The GameManager is responsible for instantiating every manager and prefabed game object during runtime. Here you set most of the framework setting directly, or you set other game object that have settings in their inspector windows.



Player setup: Here is where you set the player prefab, as well as the number of players and the different colors to mark the player(you need as many colors as players not to get errors).
Map setup: Here you pick whether you want to use a randomly generated or a premade map, set the prefabs of them, and turn on or off spawning resources and neutral units for randomly generated maps.
Turn setup: Just drag the turn manager here, it contains all the settings in it's inspector.
Highlight setup: Y ou can pick if you want to use the highlight system, and the prefabs for the various markers.
UI setup: Just drag the UI manager prefab here.

The GameManager object also has the InputManager script component. Most of the players input is done by clicking around on the map and the UI, but you also have the option to use hotkeys for some actions(this is really useful for gamepad input). You can change the key bindings in the InputManager script attached to the GameManager gameobject, using KeyCodes that bypass your projects input manager settings,



or using a string that corresponds to a name of an axis in your input manager (this way the player can change the key bindings from the unity launcher settings)

VIII. Future of the framework

1. Wishlist

This is a list of functionalities that will be added to the framework. If you need this framework to implement another functionality, want to tell me to make one of the listed ones a higher priority, or want to help develop one of them, please don't refrain from contacting me.

A. Vertex snapping for premade map tile editing
B. Generate map to prefab for further editing
C. Gamepad on-map input

2. Known bugs

This is a list of known bugs. If you find another bug, want to help fix a bug, or need help fixing it cause you can't wait for me to fix it, please don't refrain from contacting me.

A. Highlight system for safe/dangerous movement tiles doesn't work properly for hex tiles on premade maps

Contact: simplevideogamesdev@gmail.com