SuperTiles

Documentation

Online-documentation

Join our Discord

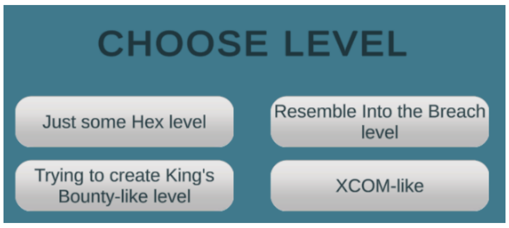
Contents

Join our Discord

Overview

This asset is developed to speed up creation of turn-based games, mostly 3D games. (2D games is direction for development in the near future)

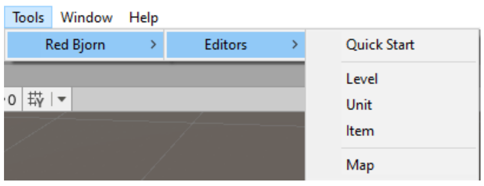
To have a better representation of asset functionality you could check several examples of levels, by loading Menu scene:


CHOOSE LEVEL
Just some Hex level
Resemble Into the Breach level
Trying to create King's Bounty-like level
XCOM-like

- Some of them have square tile grid and some have hex
- With squad or move range turn rule.

SuperTiles has slightly more than 200 C# scripts, but as it is based on the other asset – ProtoTiles which helps in tile map creation, it consists of 140 C# own scripts. Even it looks like a lot, most of them match the Single Responsibility Principle, and average row count of a single script is small enough. *SuperTiles* code is fully available: no dll or any other libraries. Key points are presented as interfaces or abstract classes to give you a power of easy modification to comply with game features which have not been implemented yet in this asset.

Also, the asset contains a bunch of Editor windows for simplification the process of main stuff creation (items, units, levels). They could be found under Tools/Red Bjorn/Editors submenu



The main purpose of *SuperTiles* is to become the most helpful asset for creation turn-based games, so it will not stay rigid and will get many updates. To achieve this your feedback is highly needed. Feel free to contact at:

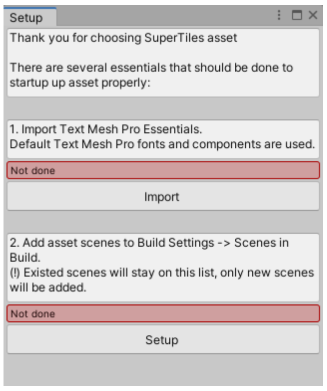
https://discord.gg/5dTM9SfqME

chatme@posteo.net

Join our Discord

Setup

After asset import, Setup window will appear.



Setup

Thank you for choosing SuperTiles asset

There are several essentials that should be done to startup up asset properly:

1. Import Text Mesh Pro Essentials.
Default Text Mesh Pro fonts and components are used.

Not done

Import

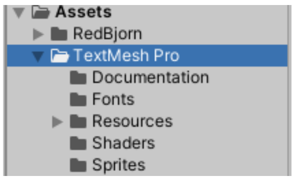
2. Add asset scenes to Build Settings -> Scenes in Build.
(!) Existed scenes will stay on this list, only new scenes will be added.

Not done

Setup

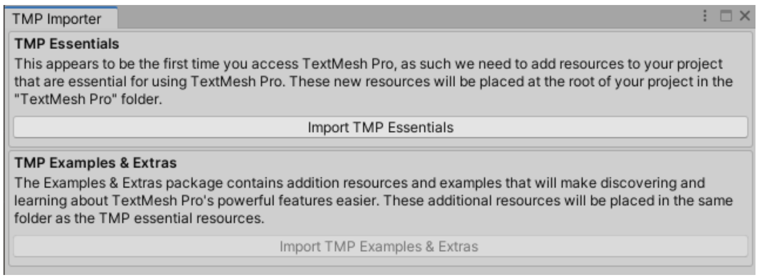
There are two rules which will be checked:

- Does project contain Text Mesh Pro essentials?



If “Not done” label appear, you could fix it:

1. Click Import button to open TMP Importer window



TMP Importer

TMP Essentials

This appears to be the first time you access TextMesh Pro, as such we need to add resources to your project that are essential for using TextMesh Pro. These new resources will be placed at the root of your project in the "TextMesh Pro" folder.

Import TMP Essentials

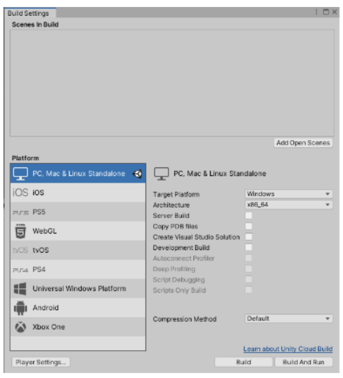
TMP Examples & Extras

The Examples & Extras package contains addition resources and examples that will make discovering and learning about TextMesh Pro's powerful features easier. These additional resources will be placed in the same folder as the TMP essential resources.

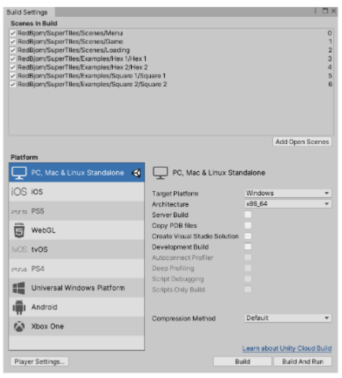
Import TMP Examples & Extras

2. Click Import TMP Essentials.

- Does Scene in Build list contain essential scenes?
  If “Not done” label appear, you could fix it, by clicking Setup button



Target Platform: Windows
Architecture: x86_64
Server Build
Copy PDB files
Create Visual Studio Solution
Development Build
Autoconnect Profiler
Deep Profiling
Script Debugging
Scripts Only Build
Compression Method: Default



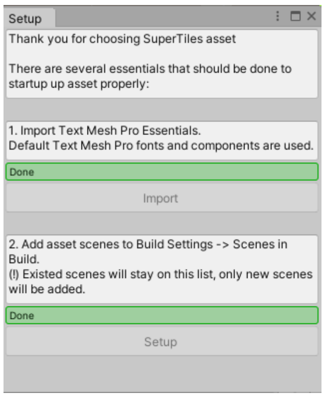
Build Settings

Scenes In Build

✓ RedBjorn/SuperTiles/Scenes/Menu	0
✓ RedBjorn/SuperTiles/Scenes/Game	1
✓ RedBjorn/SuperTiles/Scenes/Loading	2
✓ RedBjorn/SuperTiles/Examples/Hex 1/Hex 1	3
✓ RedBjorn/SuperTiles/Examples/Hex 2/Hex 2	4
✓ RedBjorn/SuperTiles/Examples/Square 1/Square 1	5
✓ RedBjorn/SuperTiles/Examples/Square 2/Square 2	6

Join our Discord

When all setup is done correctly Setup window become all “green”


Setup

Thank you for choosing SuperTiles asset

There are several essentials that should be done to startup up asset properly:

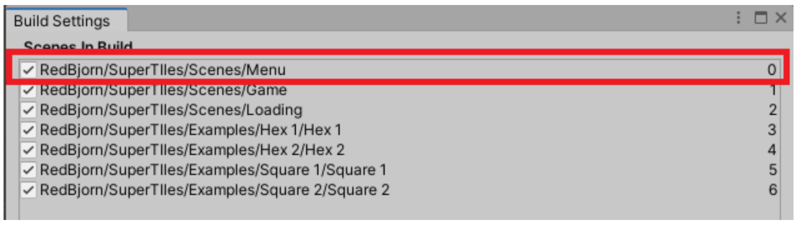
1. Import Text Mesh Pro Essentials.
Default Text Mesh Pro fonts and components are used.

2. Add asset scenes to Build Settings -> Scenes in Build.
(!) Existed scenes will stay on this list, only new scenes will be added.

Join our Discord

How to run the asset

First, you should load Menu scene. It is the initial scene for the asset.

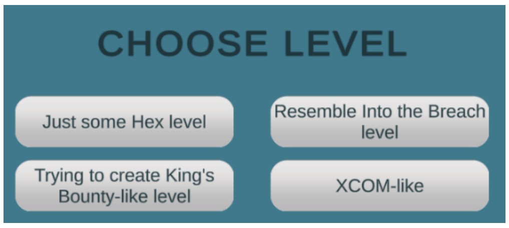


Build Settings

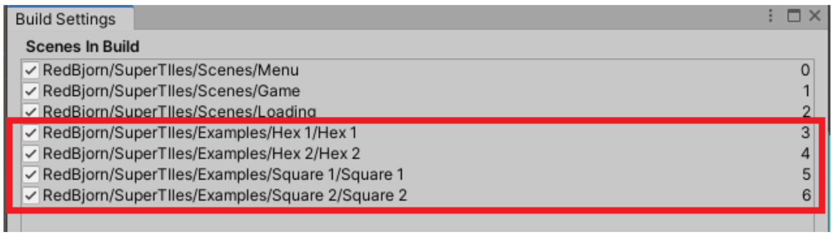
Scenes In Build

✓ RedBjorn/SuperTIles/Scenes/Menu	0
✓ RedBjorn/SuperTIles/Scenes/Game	1
✓ RedBjorn/SuperTIles/Scenes/Loading	2
✓ RedBjorn/SuperTIles/Examples/Hex 1/Hex 1	3
✓ RedBjorn/SuperTIles/Examples/Hex 2/Hex 2	4
✓ RedBjorn/SuperTIles/Examples/Square 1/Square 1	5
✓ RedBjorn/SuperTIles/Examples/Square 2/Square 2	6

Here you can see 4 level load buttons. Click any of them and game will load immediately


CHOOSE LEVEL
Just some Hex level
Resemble Into the Breach level
Trying to create King's Bounty-like level
XCOM-like

Also, you could start from any of the example scenes. All of them have redirection to Menu scene



Build Settings

Scenes In Build

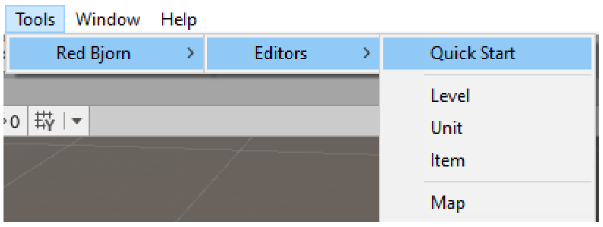
✓ RedBjorn/SuperTIles/Scenes/Menu	0
✓ RedBjorn/SuperTIles/Scenes/Game	1
✓ RedBjorn/SuperTIles/Scenes/Loading	2
✓ RedBjorn/SuperTIles/Examples/Hex 1/Hex 1	3
✓ RedBjorn/SuperTIles/Examples/Hex 2/Hex 2	4
✓ RedBjorn/SuperTIles/Examples/Square 1/Square 1	5
✓ RedBjorn/SuperTIles/Examples/Square 2/Square 2	6

The rest 2 scenes: Game and Loading are helper scenes. Loading from them will not lead you to a proper game state.

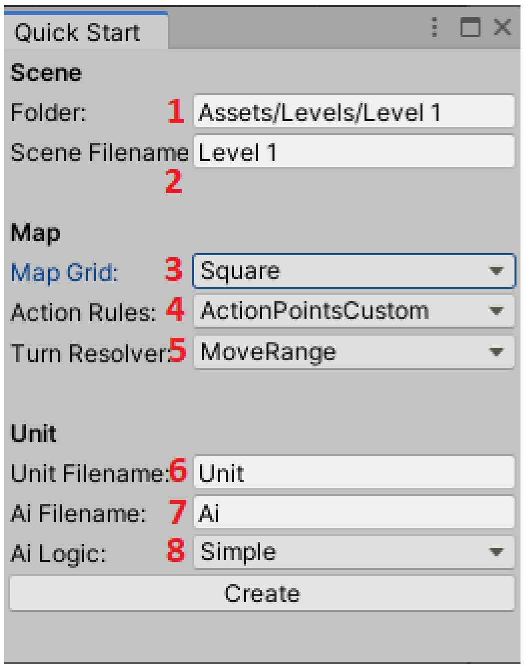
Join our Discord

Quick Start

To create your own level, select Quick start submenu item.



You will see Quick Start window with already filled fields.



Quick Start

Scene

Folder: 1 Assets/Levels/Level 1

Scene Filename 2 Level 1

Map

Map Grid: 3 Square

Action Rules: 4 ActionPointsCustom

Turn Resolver 5 MoveRange

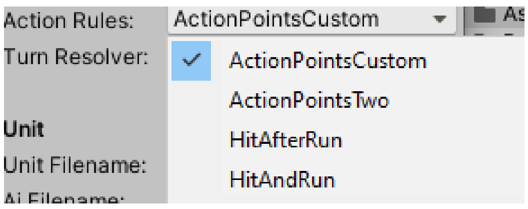
Unit

Unit Filename 6 Unit

Ai Filename: 7 Ai

Ai Logic: 8 Simple

1. Folder is the root folder of level related assets
2. Scene Filename is the name of scene file without .unity extension
3. Map Grid could be Square or Hex.
4. Action Rules define the way unit could male it's actions
5. Turn Resolver defines the way turn completion is handled
6. Unit Filename is the name of asset file which will store information about unit (at least one unit should be created to start the level)
7. Ai Filename is the name of asset file which will store information about unit artificial intelligence
8. Ai Logic is the type of Ai behavior. Right now, there is only 1 type: SimpleLogic



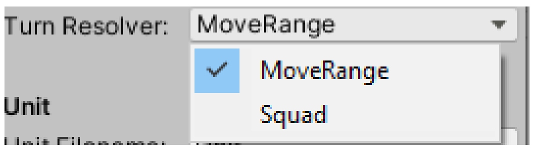
ActionPoinstCustom allow to spend X Points (2 by default), where 1 move action cost 1 Point and 1 item action cost also 1 Point. Move action after Item action is prohibited.

ActionPoinstTwo = ActionPoinstCustom where X = 2 constanlty (most Common)

HitAfterRun allows

- 1 move action or
- 1 item action or
- 1 item action after 1 move action

HitAndRun allows 1 move action and 1 item action in any order

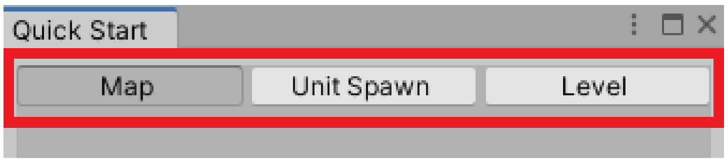


MoveRange: after the unit will do its' actions the turn will be moved to the next unit (it can be in other squad) in list ordered by move range value.

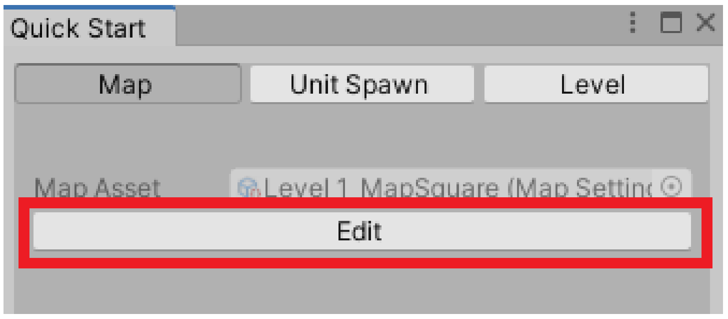
Squad: all units in squad could do its' actions and then turn will be moved to other squad

[Join our Discord](#)

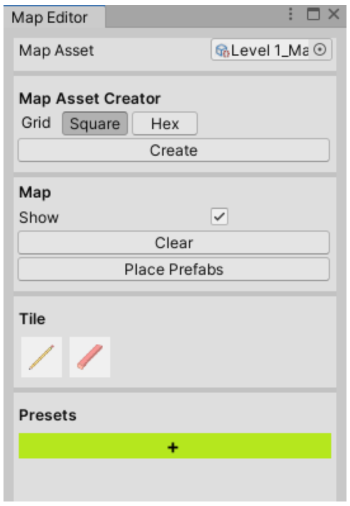
After specifying the desired values and clicking Create button you will see next three steps



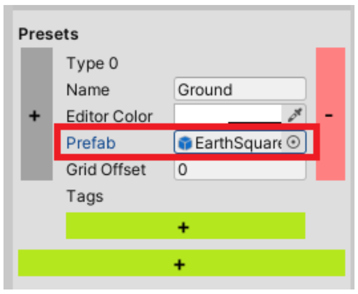
Let us start from map creation like it is conceived and click Edit button



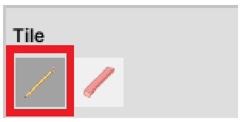
You will see Map Editor window (more information about Map Editor is inside [ProtoTiles documentation]())



Click Plus button to add initial tile preset and specify prefab field with the cube of 1 unit height and width, for example, EarthSquare

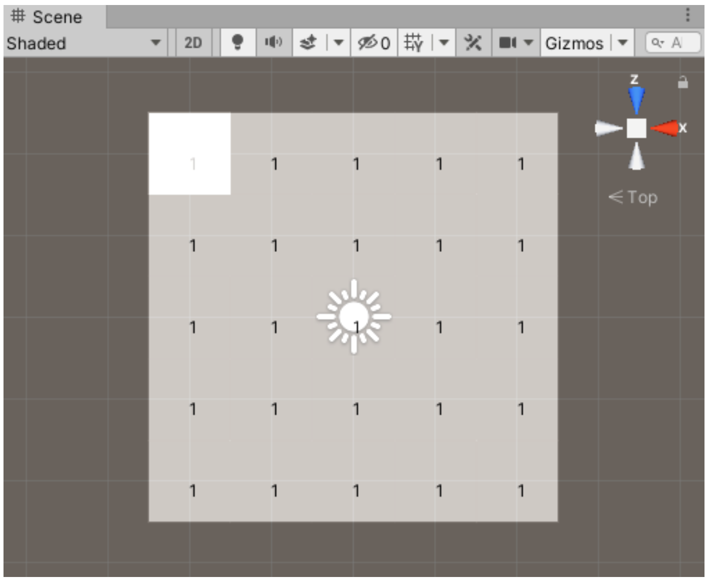


Then click Pencil button to draw the map at the Scene view tab.

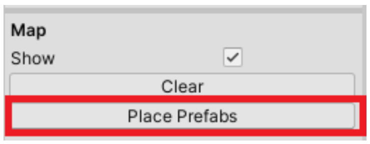


[Join our Discord]()

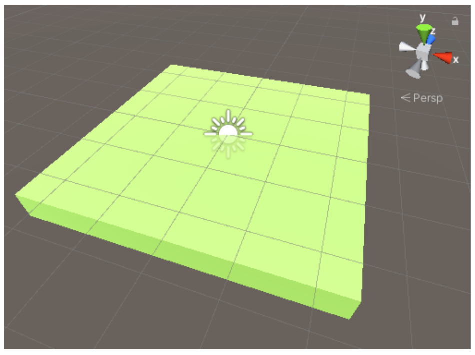
Let us draw 5x5 tile map (or any map you can imagine)



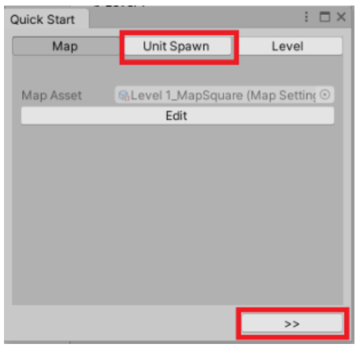
Click Place Prefabs button and close Map Editor


Map
Show ✓
Clear
Place Prefabs

You will see created map at the Scene View

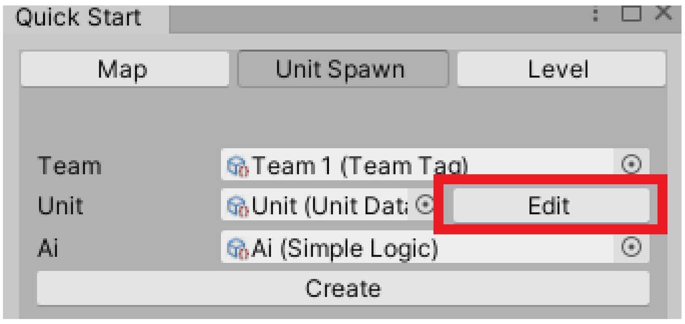


Go to the Quick Start window and move to the next step - UnitSpawn


Quick Start
Map | Unit Spawn | Level
Map Asset: Level 1_MapSquare (Map Setting
Edit
>>

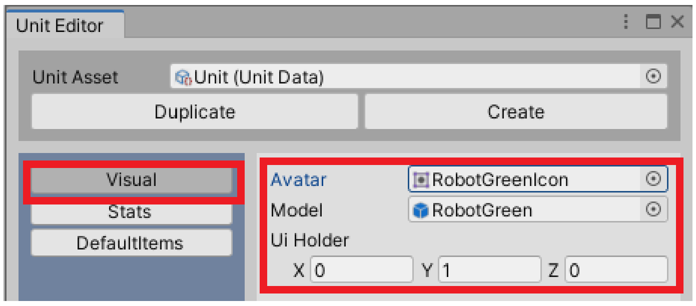
Join our Discord

Here you will see already filled fields for Unit spawn point creation. But do not hurry, first Click Edit button to configure our blank Unit asset. It will be the template for other units.



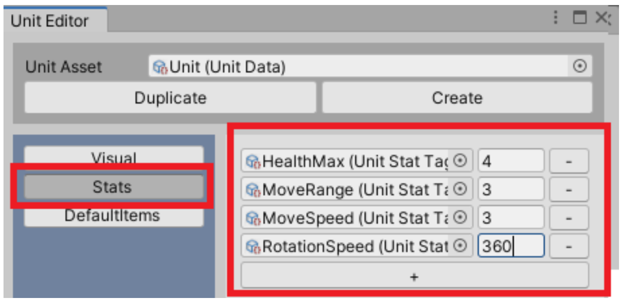
Team: Team 1 (Team Tag)
Unit: Unit (Unit Dat: | Edit
Ai: Ai (Simple Logic)

New window (Unit Editor) will open. Define Visual parameters



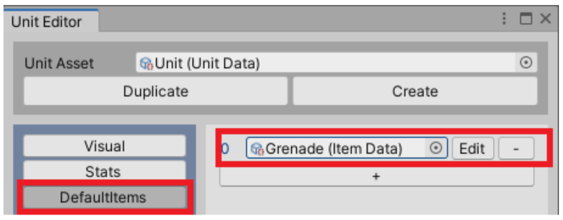
Unit Asset: Unit (Unit Data)
Avatar: RobotGreenIcon
Model: RobotGreen
Ui Holder
X 0 Y 1 Z 0

Then define Stats parameters



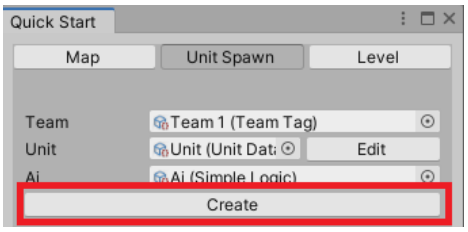
Unit Asset: Unit (Unit Data)
HealthMax (Unit Stat Taç 4
MoveRange (Unit Stat T: 3
MoveSpeed (Unit Stat T: 3
RotationSpeed (Unit Stat 360

And add at least one default item. You could also create new item here, clicking Edit button will open Item editor, but for now, let us choose from already created items, for example, Grenade and return to Quick Start window.



Unit Asset: Unit (Unit Data)
0 Grenade (Item Data)

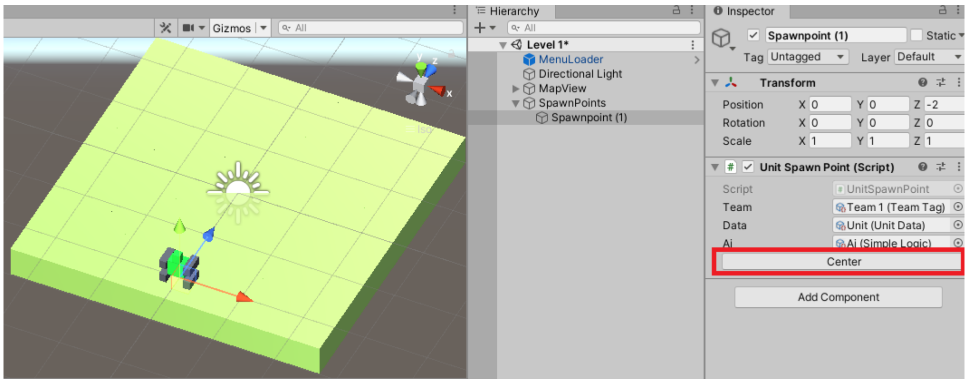
Click Create button to add Unit Spawn point to the Scene View tab



Team: Team 1 (Team Tag)
Unit: Unit (Unit Dat:
Ai: Ai (Simple Logic)

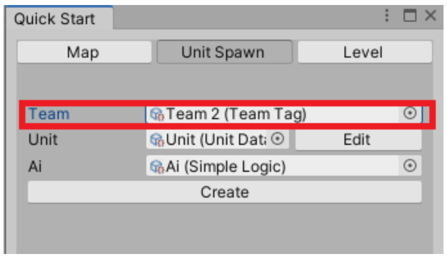
Join our Discord

Spawnpoint (1) will be created at Vector3.zero position and you can move it to any map position and click Center button to align with the tile center

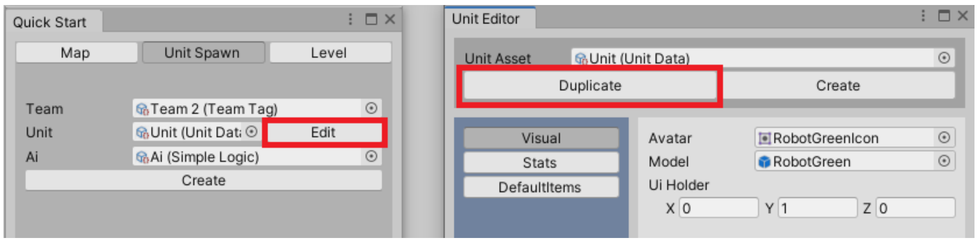

Hierarchy
Level 1*
MenuLoader
Directional Light
MapView
SpawnPoints
Spawnpoint (1)

Inspector
Spawnpoint (1) Static
Tag Untagged Layer Default
Transform
Position X 0 Y 0 Z -2
Rotation X 0 Y 0 Z 0
Scale X 1 Y 1 Z 1
Unit Spawn Point (Script)
Script UnitSpawnPoint
Team Team 1 (Team Tag)
Data Unit (Unit Data)
Ai Ai (Simple Logic)
Center
Add Component

Return to Quick Start window and change unit team to Team 2

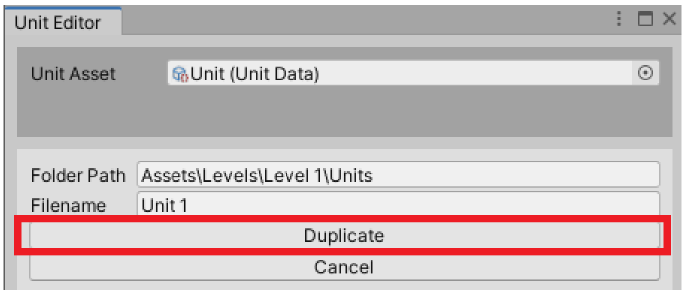

Quick Start
Map | Unit Spawn | Level
Team: Team 2 (Team Tag)
Unit: Unit (Unit Data) Edit
Ai: Ai (Simple Logic)
Create

Let us create another unit to prevent enemy unit look like our unit. Click Edit button than Duplicate button at Unit Editor


Quick Start
Map | Unit Spawn | Level
Team: Team 2 (Team Tag)
Unit: Unit (Unit Data) Edit
Ai: Ai (Simple Logic)
Create

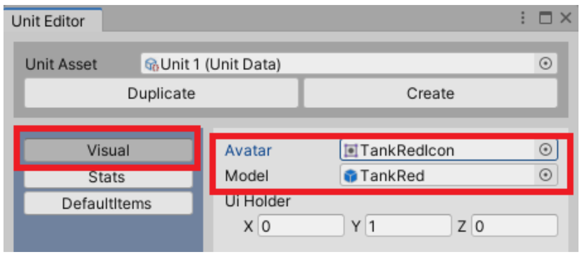
Unit Editor
Unit Asset: Unit (Unit Data)
Duplicate | Create
Visual | Stats | DefaultItems
Avatar: RobotGreenIcon
Model: RobotGreen
Ui Holder: X 0 Y 1 Z 0

After specifying desired filename click Duplicate button.


Unit Editor
Unit Asset: Unit (Unit Data)
Folder Path: Assets\Levels\Level 1\Units
Filename: Unit 1
Duplicate
Cancel

[Join our Discord](https://discord.com)

After asset creation, new unit will be selected at Unit Editor window. Let us change its model.


Unit Editor

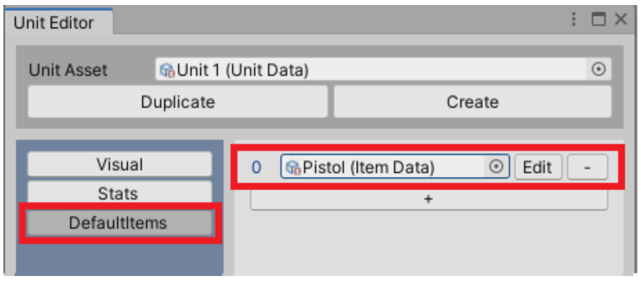
Unit Asset: Unit 1 (Unit Data)

Duplicate | Create

Visual
Stats
DefaultItems

Avatar: TankRedIcon
Model: TankRed
Ui Holder
X 0 Y 1 Z 0

And change its default item


Unit Editor

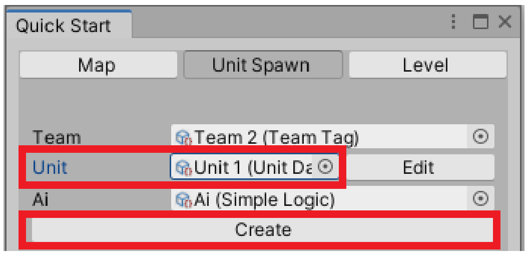
Unit Asset: Unit 1 (Unit Data)

Duplicate | Create

Visual
Stats
DefaultItems

0 | Pistol (Item Data) | Edit | -
+

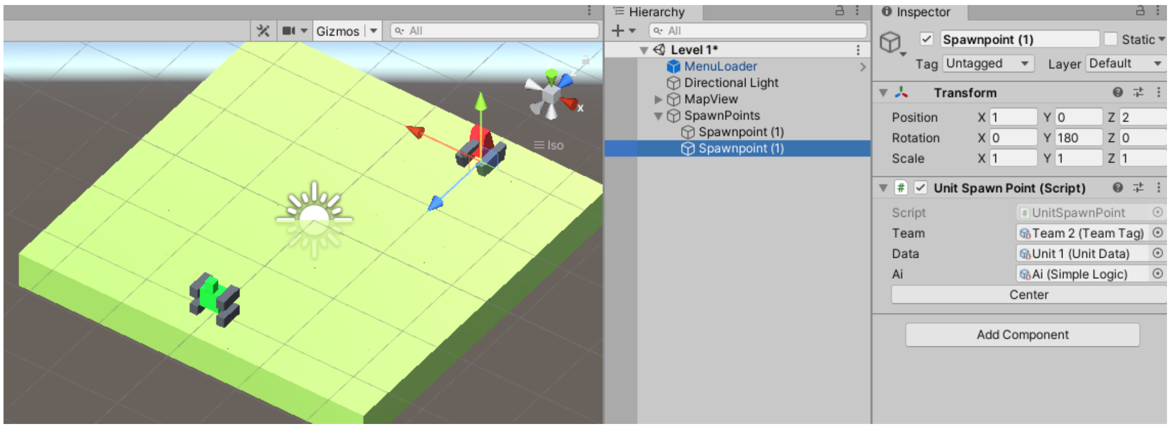
Return to Quick Start window and define new unit inside Unit field and click Create button


Quick Start

Map | Unit Spawn | Level

Team: Team 2 (Team Tag)
Unit: Unit 1 (Unit Da | Edit
Ai: Ai (Simple Logic)
Create

Move new unit to correct position. Do not forget about Center button to align spawnpoint position



Hierarchy
- Level 1*
  - MenuLoader
  - Directional Light
  - MapView
  - SpawnPoints
    - Spawnpoint (1)
    - Spawnpoint (1)

Inspector
Spawnpoint (1) | Static
Tag Untagged | Layer Default

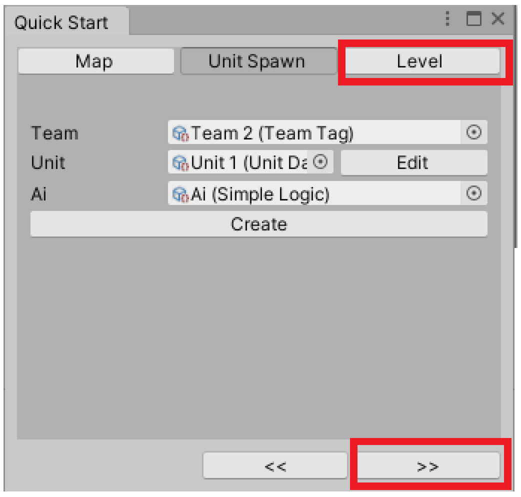
Transform
Position X 1 Y 0 Z 2
Rotation X 0 Y 180 Z 0
Scale X 1 Y 1 Z 1

Unit Spawn Point (Script)
Script: UnitSpawnPoint
Team: Team 2 (Team Tag)
Data: Unit 1 (Unit Data)
Ai: Ai (Simple Logic)
Center

Add Component

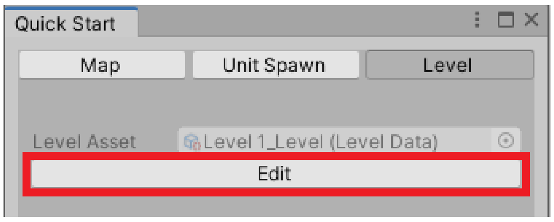
[Join our Discord](#)

In Quick Start window move to the next step – Level



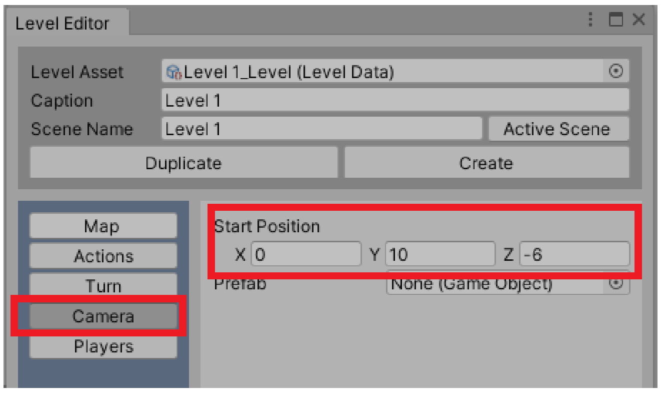
Team: Team 2 (Team Tag)
Unit: Unit 1 (Unit Da
Ai: Ai (Simple Logic)

Click Edit button to open Level Editor window



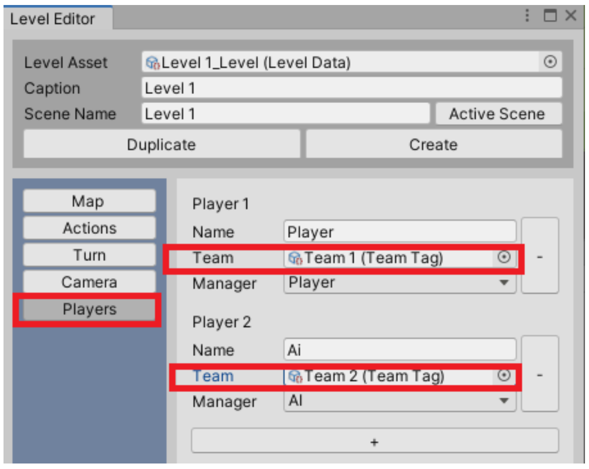
Level Asset: Level 1_Level (Level Data)

At the Camera tab specify Start Position. For our map (0, 10, -6) will be appropriate position.



Level Asset: Level 1_Level (Level Data)
Caption: Level 1
Scene Name: Level 1
Start Position
X 0 Y 10 Z -6
Prefab: None (Game Object)

At the Players tab specify team which will be controlled by player and which by ai



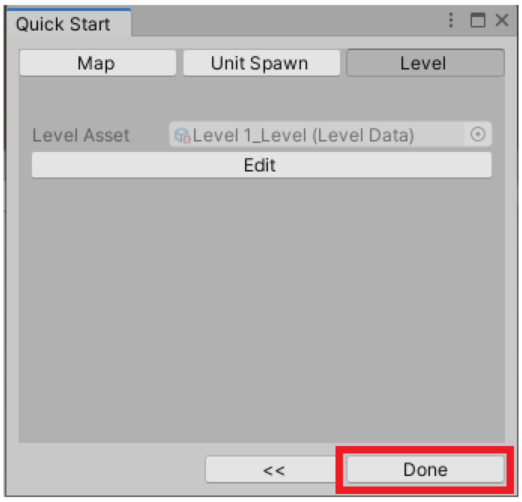
Level Asset: Level 1_Level (Level Data)
Caption: Level 1
Scene Name: Level 1

Player 1
Name: Player
Team: Team 1 (Team Tag)
Manager: Player

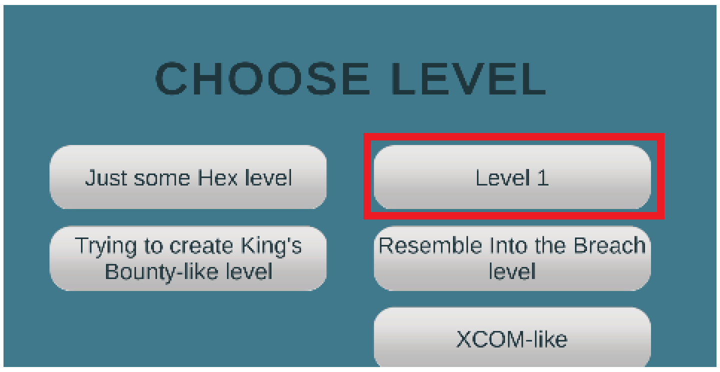
Player 2
Name: Ai
Team: Team 2 (Team Tag)
Manager: AI

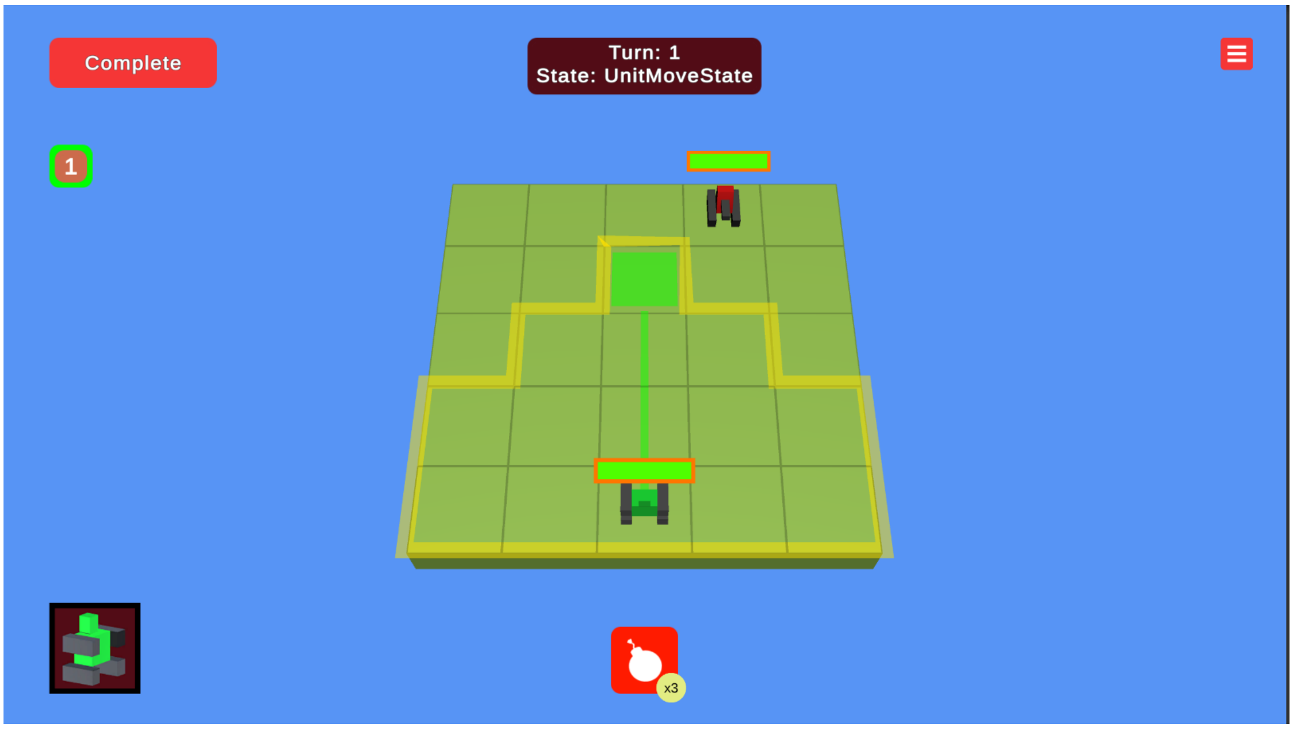
Join our Discord

Return to Quick Start window and click Done button



Ta-dam! Congratulations you have created your own level. To check it, click Playmode button. You will be redirected to Menu scene.


CHOOSE LEVEL
Just some Hex level
Level 1
Trying to create King's Bounty-like level
Resemble Into the Breach level
XCOM-like



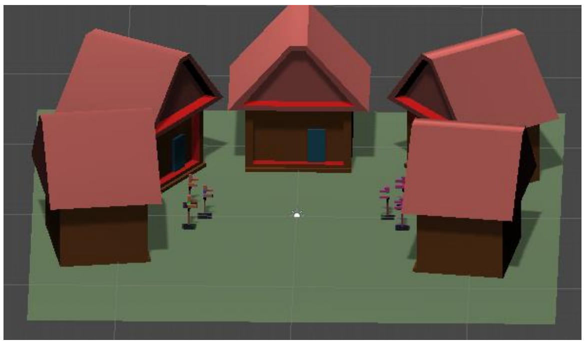
[Join our Discord](https://discord.gg)

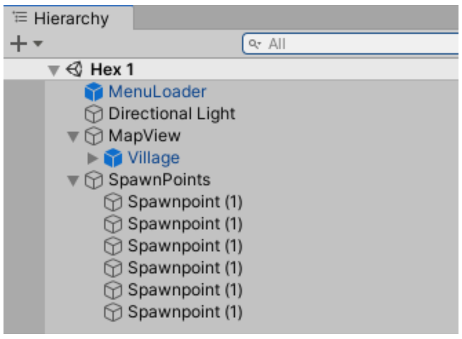
Concept

As usual game, *SuperTiles* uses a bunch of Data and Logic which handles its Data.

For every important part of Data and Logic there is individual section which contains more detail description. Here it will be explained only the basis of *SuperTiles*:

On the bottom of our Data concept there is a well-known unity scene which contain visual information about the level. Map representation is located under MapView gameObject (in our example it is Village prefab).

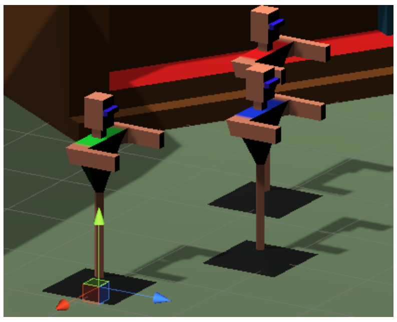


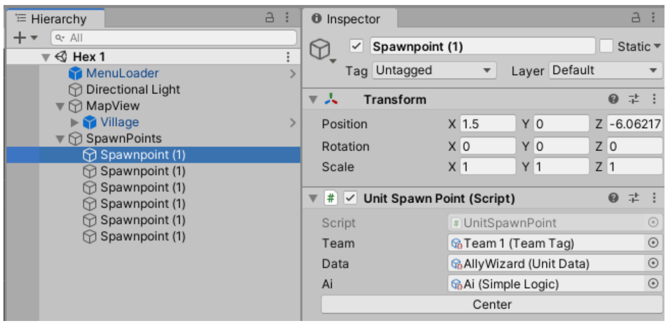


Hierarchy

```
▼ Hex 1
    MenuLoader
    Directional Light
  ▼ MapView
    ▶ Village
  ▼ SpawnPoints
      Spawnpoint (1)
      Spawnpoint (1)
      Spawnpoint (1)
      Spawnpoint (1)
      Spawnpoint (1)
      Spawnpoint (1)
```

Also, scene contains the data of units inside spawnpoint gameObjects. UnitSpawnPoint class define which Team the unit will belong, what kind of unit (Data field) will be spawned at the corresponding position and what UnitAiData will be used if it will be controlled by Ai player





Hierarchy

```
▼ Hex 1
    MenuLoader
    Directional Light
  ▼ MapView
    ▶ Village
  ▼ SpawnPoints
      Spawnpoint (1)
      Spawnpoint (1)
      Spawnpoint (1)
      Spawnpoint (1)
      Spawnpoint (1)
      Spawnpoint (1)
```

Inspector

Spawnpoint (1) — Static — Tag: Untagged — Layer: Default

Transform

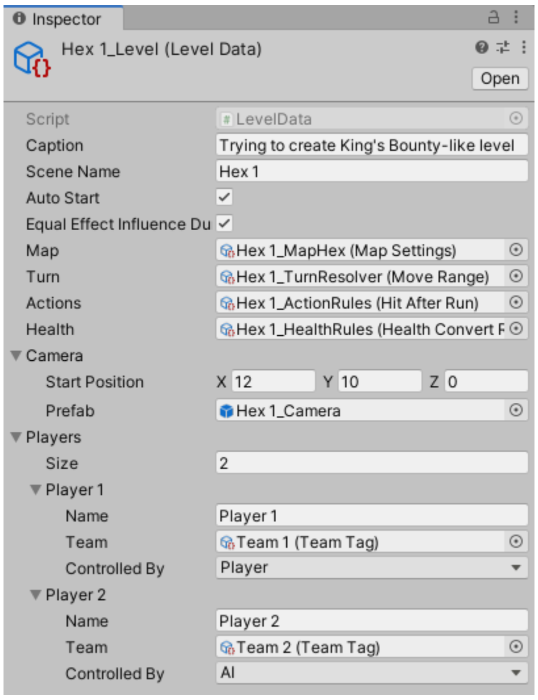
	X	Y	Z
Position	1.5	0	-6.06217
Rotation	0	0	0
Scale	1	1	1

Unit Spawn Point (Script)

Script	UnitSpawnPoint
Team	Team 1 (Team Tag)
Data	AllyWizard (Unit Data)
Ai	Ai (Simple Logic)

Center

Futher, the reference to this scene is stored inside LevelData, which is another key point of the concept.



Inspector

Hex 1_Level (Level Data) — Open

Script	LevelData
Caption	Trying to create King's Bounty-like level
Scene Name	Hex 1
Auto Start	✓
Equal Effect Influence Du	✓
Map	Hex 1_MapHex (Map Settings)
Turn	Hex 1_TurnResolver (Move Range)
Actions	Hex 1_ActionRules (Hit After Run)
Health	Hex 1_HealthRules (Health Convert F
▼ Camera	
Start Position	X 12 Y 10 Z 0
Prefab	Hex 1_Camera
▼ Players	
Size	2
▼ Player 1	
Name	Player 1
Team	Team 1 (Team Tag)
Controlled By	Player
▼ Player 2	
Name	Player 2
Team	Team 2 (Team Tag)
Controlled By	AI

Join our Discord

Apart from the scene reference (as Scene Name field), LevelData keeps MapSettings link which define what grid settings will be used at this scene (which tiles are movable and which are not).

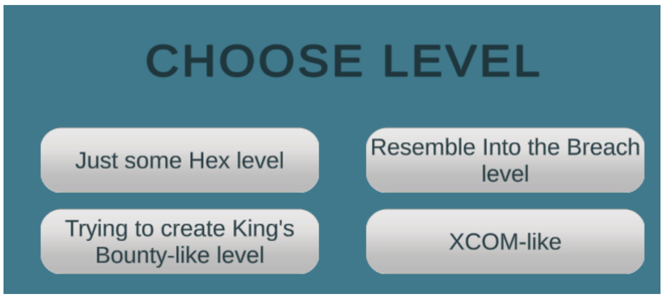
Also, LevelData defines the way the turn is moving from one unit to another, by specifying Turn field.

Actions field is neccesary for counting how many move and item actions can be done by a single unit and describes the valid order of this actions.

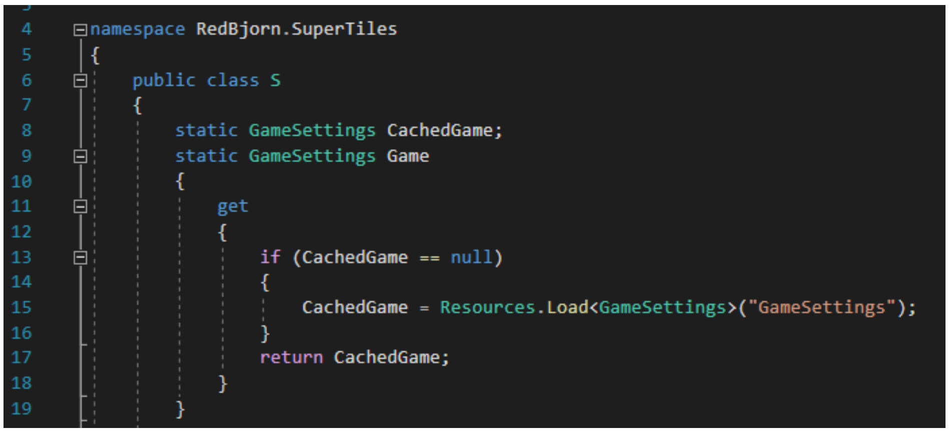
Besides, we need to know who will controll every team. This information is also a part of LevelData and can be found at the Players field.

After a brief Data review, let's move to Logic description

From above, it is known that Menu is initial scene to be loaded, but there is no direct reference to any LevelData. So where it is?

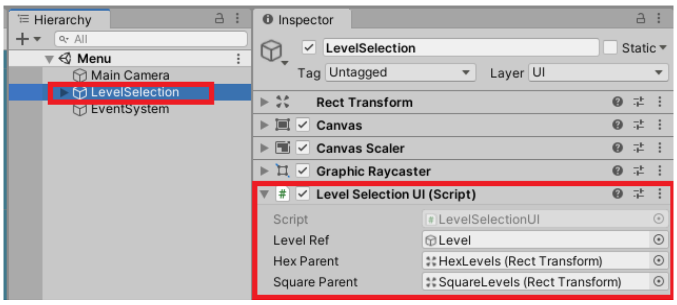

CHOOSE LEVEL
Just some Hex level
Resemble Into the Breach level
Trying to create King's Bounty-like level
XCOM-like

It worth to mention that we have something like root Data object called GameSettings. It is located inside Resource folder and have lazy initialization.



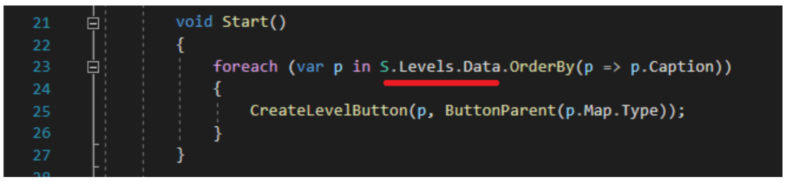
```
namespace RedBjorn.SuperTiles
{
    public class S
    {
        static GameSettings CachedGame;
        static GameSettings Game
        {
            get
            {
                if (CachedGame == null)
                {
                    CachedGame = Resources.Load<GameSettings>("GameSettings");
                }
                return CachedGame;
            }
        }
```

Exactly such reference is used inside Menu Scene at LevelSelectionUI script.



Hierarchy
Menu
Main Camera
LevelSelection
EventSystem

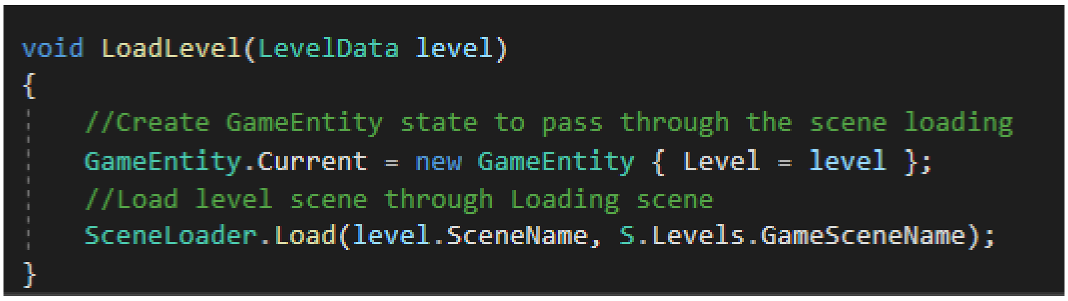
Inspector
LevelSelection Static
Tag Untagged Layer UI
Rect Transform
Canvas
Canvas Scaler
Graphic Raycaster
Level Selection UI (Script)
Script LevelSelectionUI
Level Ref Level
Hex Parent HexLevels (Rect Transform)
Square Parent SquareLevels (Rect Transform)



```
        void Start()
        {
            foreach (var p in S.Levels.Data.OrderBy(p => p.Caption))
            {
                CreateLevelButton(p, ButtonParent(p.Map.Type));
            }
        }
```

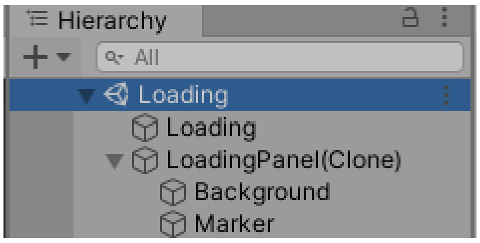
Join our Discord

Click on any level button register the level index inside LeveEntity class and start the process of level loading



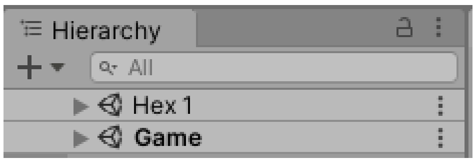
```
void LoadLevel(LevelData level)
{
    //Create GameEntity state to pass through the scene loading
    GameEntity.Current = new GameEntity { Level = level };
    //Load level scene through Loading scene
    SceneLoader.Load(level.SceneName, S.Levels.GameSceneName);
}
```

Level loading is performed by transitioning to almost empty Loading scene to prevent ungly render stuck when fat scene file should be loaded. It was said “almost”, because there are objects representing loading process



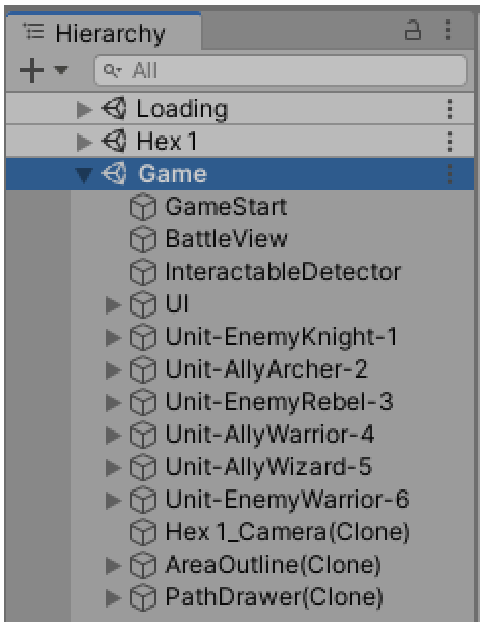
```
Hierarchy
All
▼ Loading
    Loading
  ▼ LoadingPanel(Clone)
      Background
      Marker
```

After level will be loaded, it is needed to add common stuff (UI, Main camera etc.) And another helper scene (called Game) will be loaded.



```
Hierarchy
All
▶ Hex 1
▶ Game
```

Game scene contains the starter script called GameStart where essential classes are created. Ussually essential classes will contain Entity suffix (GameEntity, UnitEntity, ItemEntity etc.). It indicates that

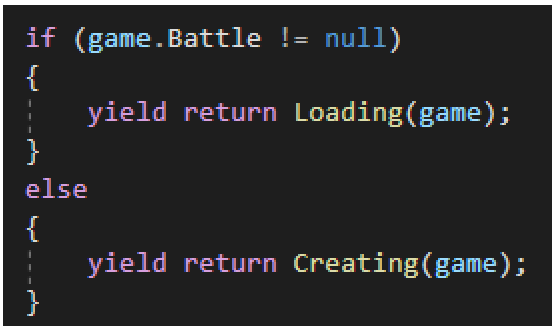


```
Hierarchy
All
▶ Loading
▶ Hex 1
▼ Game
    GameStart
    BattleView
    InteractableDetector
  ▶ UI
  ▶ Unit-EnemyKnight-1
  ▶ Unit-AllyArcher-2
  ▶ Unit-EnemyRebel-3
  ▶ Unit-AllyWarrior-4
  ▶ Unit-AllyWizard-5
  ▶ Unit-EnemyWarrior-6
    Hex 1_Camera(Clone)
  ▶ AreaOutline(Clone)
  ▶ PathDrawer(Clone)
```

Join our Discord

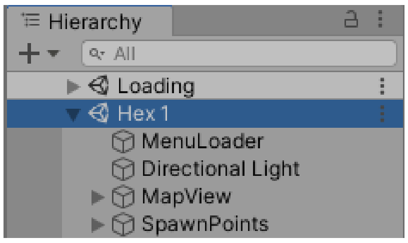
these classes are plain c# classes and could be easyly serialized. The main essential class is GameEntity. It stores full state of the current level and also is used to recreate game from savefile.

During the level loading procces *GameEntity* defines should be the game created as new session or need to be loaded from existed state. To undestand what path is right we check if *GameEntity* contains something in *Battle* field.

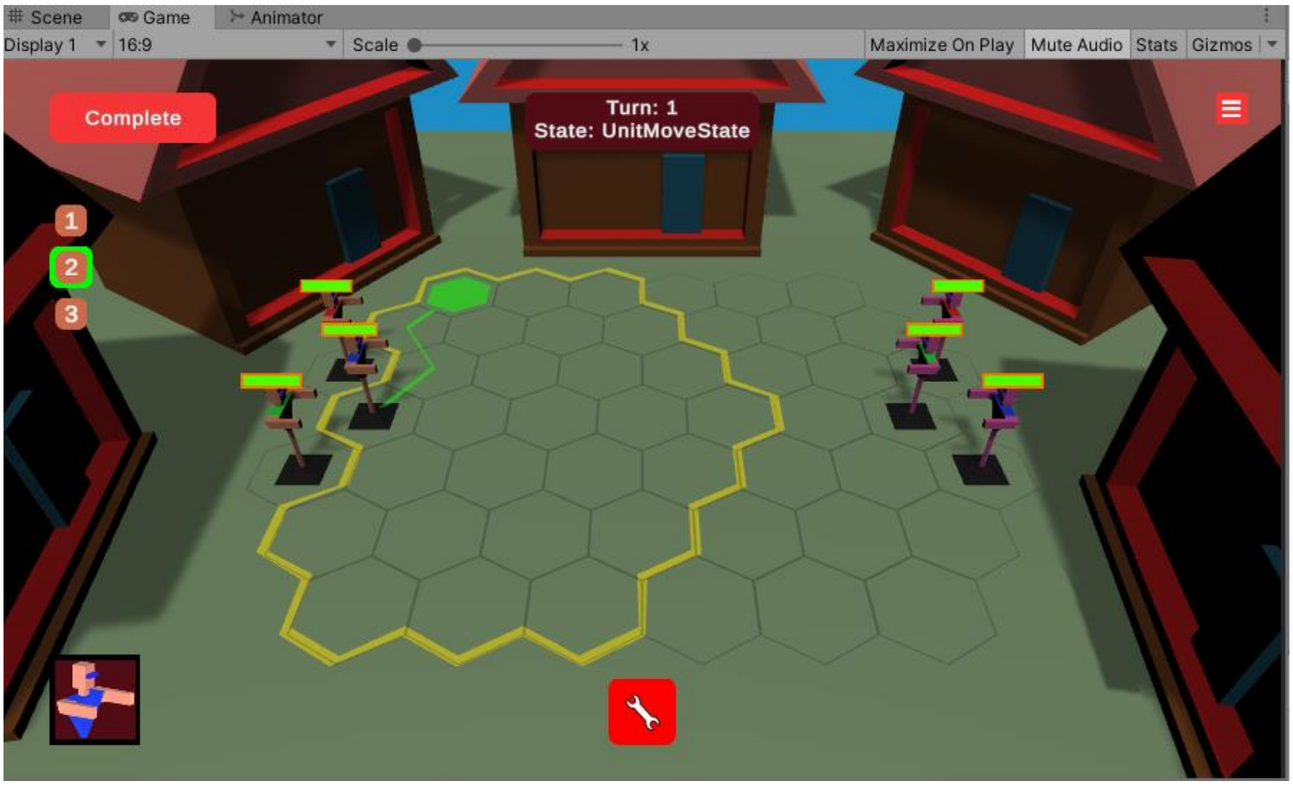


```
if (game.Battle != null)
{
    yield return Loading(game);
}
else
{
    yield return Creating(game);
}
```

When load process is done, Loading scene will be unloaded and you can start playing the game. Playing the game means that you start a Battle, and the description of this process is declared here



```
Hierarchy
+ ▾  All
▶ Loading
▼ Hex 1
    MenuLoader
    Directional Light
  ▶ MapView
  ▶ SpawnPoints
```



Join our Discord

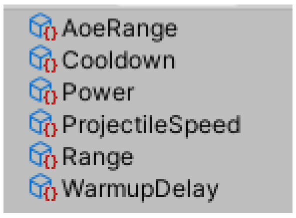
Core

Tags

Tags in *SuperTiles* differ from Unity tags, although they serve the same purpose. There this only one difference – implementation. Unit tag is a single string which is located inside TagManager asset. *SuperTiles* tag is a single ScriptableObject which is located somewhere is Assets folder. Such tag concept is not an original idea, but seems like more comfortable, because of its handy drag-and-drop native feature and flexible refactoring possibilities.

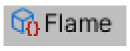
SuperTiles default tag types (successors of Tag class):

ItemStatTag



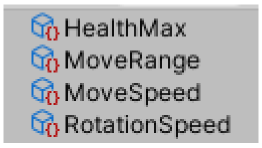
AoeRange
Cooldown
Power
ProjectileSpeed
Range
WarmupDelay

ItemTag



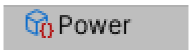
Flame

UnitStatTag



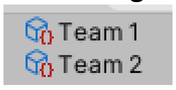
HealthMax
MoveRange
MoveSpeed
RotationSpeed

EffectStatTag



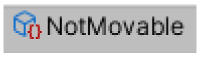
Power

TeamTag



Team 1
Team 2

TileTag (comes from ProtoTiles)



NotMovable

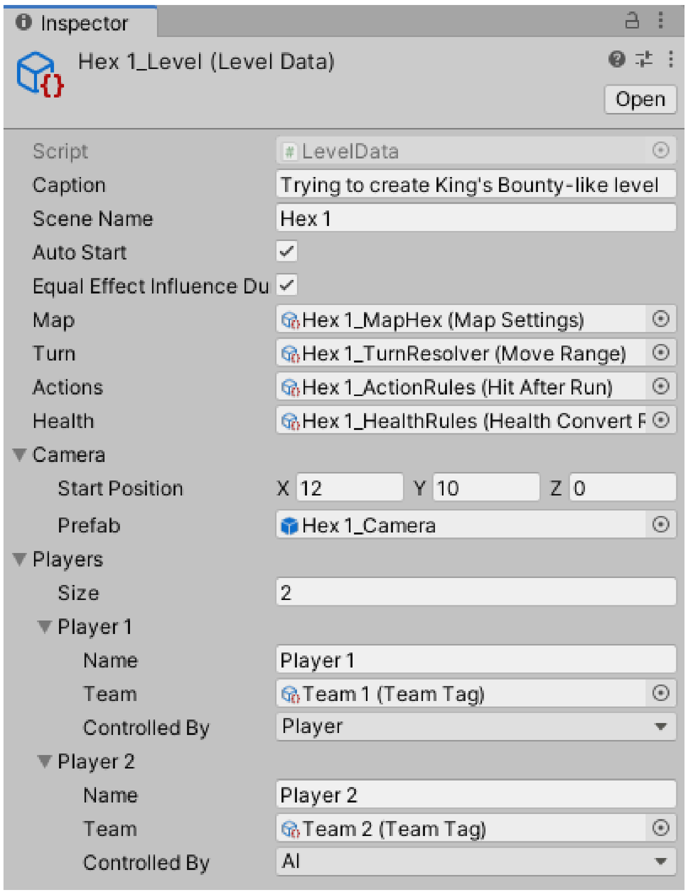
Existed tag list can be easily extended by duplicating desired instance and giving it new appropriate name. Also, you could create your own tag type by creating new Tag class successor.

```
public class TeamTag : Tag
```

Join our Discord

Level

The key point of current turn-based game implementation is LevelData class. It contains information about:



Inspector

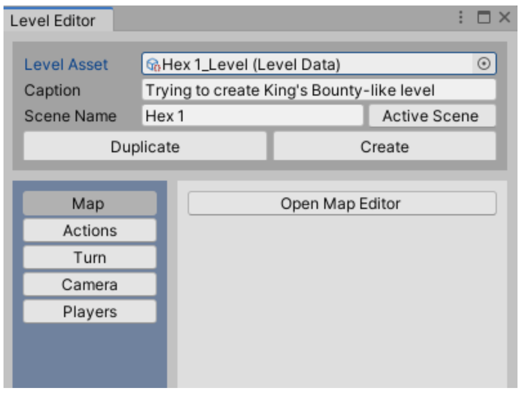
Hex 1_Level (Level Data)

Open

Script	LevelData
Caption	Trying to create King's Bounty-like level
Scene Name	Hex 1
Auto Start	✓
Equal Effect Influence Du	✓
Map	Hex 1_MapHex (Map Settings)
Turn	Hex 1_TurnResolver (Move Range)
Actions	Hex 1_ActionRules (Hit After Run)
Health	Hex 1_HealthRules (Health Convert F
Camera	
Start Position	X 12 Y 10 Z 0
Prefab	Hex 1_Camera
Players	
Size	2
Player 1	
Name	Player 1
Team	Team 1 (Team Tag)
Controlled By	Player
Player 2	
Name	Player 2
Team	Team 2 (Team Tag)
Controlled By	AI

- what Caption will pe displayed at menu
- what Scene should be loaded,
- should be battle started after scene loaded?
- should be effects stacked by duration increase?
- what MapSettings (tile grid) will be used,
- who will get the next Turn?
- which Actions rule should unit conduct during its turn?
- any custom Camera settings
- which team is controlled by Player or AI?

Level Editor window is designed for editing exactly this class.

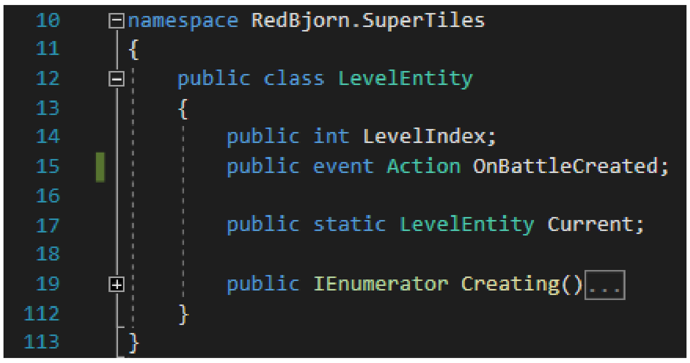


Level Editor

Level Asset	Hex 1_Level (Level Data)
Caption	Trying to create King's Bounty-like level
Scene Name	Hex 1

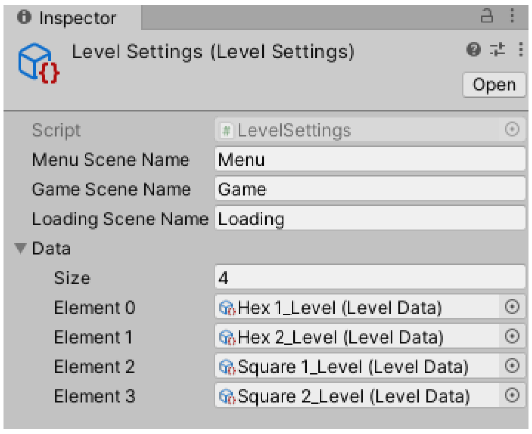
Join our Discord

Besides, level info storage (LevelData) there is another important part – LevelEntity. LevelEntity is a runtime state container, it knows information about which level is loaded and how key classes of game should be created (Creating method).



```
namespace RedBjorn.SuperTiles
{
    public class LevelEntity
    {
        public int LevelIndex;
        public event Action OnBattleCreated;

        public static LevelEntity Current;

        public IEnumerator Creating()...
    }
}
```

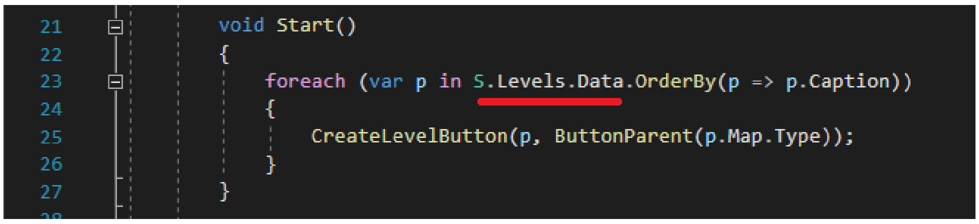
It is observed that current level is stored as integer value - LevelIndex. So, there should be a list of all LevelData somewhere. Such list is located inside LevelSettings asset.



Inspector
Level Settings (Level Settings)
Open

Script	LevelSettings
Menu Scene Name	Menu
Game Scene Name	Game
Loading Scene Name	Loading
Data	
Size	4
Element 0	Hex 1_Level (Level Data)
Element 1	Hex 2_Level (Level Data)
Element 2	Square 1_Level (Level Data)
Element 3	Square 2_Level (Level Data)

LevelSettings will be used by Menu scene to create load level buttons



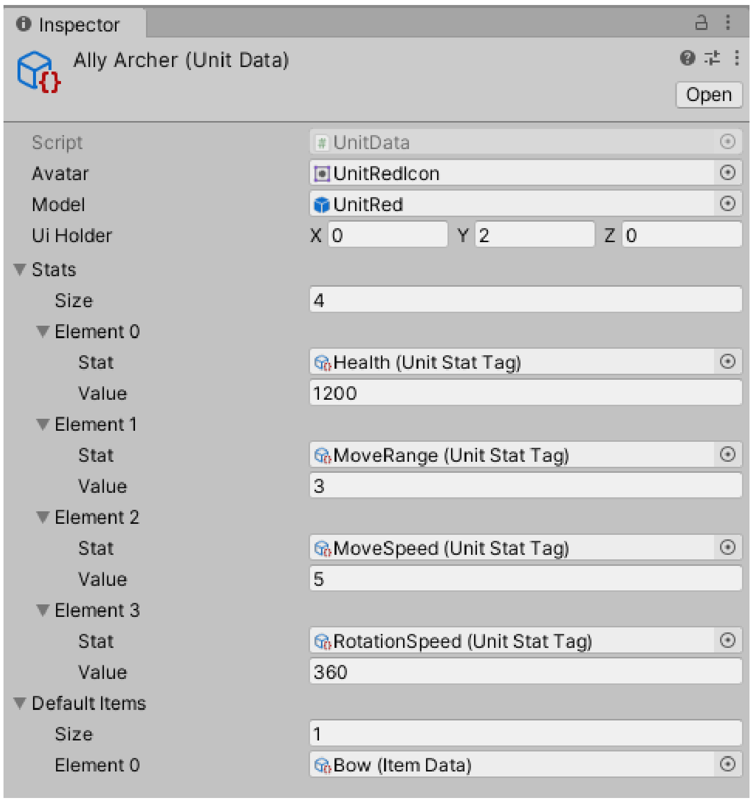
```
        void Start()
        {
            foreach (var p in S.Levels.Data.OrderBy(p => p.Caption))
            {
                CreateLevelButton(p, ButtonParent(p.Map.Type));
            }
        }
```

Join our Discord

Unit

Unit in *SuperTiles* consist of 3 main components: *UnitData*, *UnitEntity* and *UnitView*

UnitData is an information storage with default values like Health or MoveRange.



Inspector

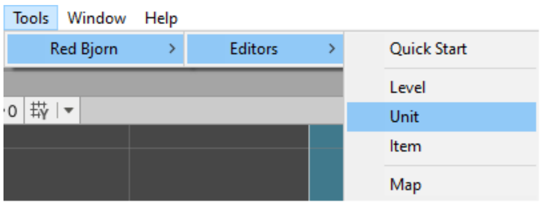
Ally Archer (Unit Data)

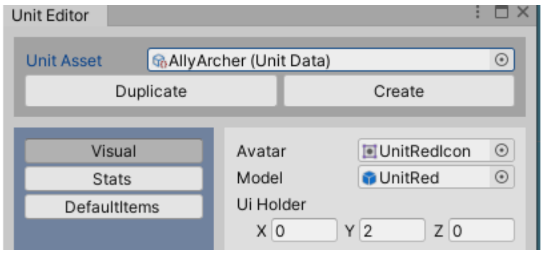
Open

Script	UnitData
Avatar	UnitRedIcon
Model	UnitRed
Ui Holder	X 0 Y 2 Z 0
▼ Stats	
Size	4
▼ Element 0	
Stat	Health (Unit Stat Tag)
Value	1200
▼ Element 1	
Stat	MoveRange (Unit Stat Tag)
Value	3
▼ Element 2	
Stat	MoveSpeed (Unit Stat Tag)
Value	5
▼ Element 3	
Stat	RotationSpeed (Unit Stat Tag)
Value	360
▼ Default Items	
Size	1
Element 0	Bow (Item Data)

Besides Stats information, *UnitData* contains visual data (Avatar, prefab Model) and list of Default items

Unit Editor window was designed to simplify Unit duplication and editing





Unit Editor

Unit Asset: AllyArcher (Unit Data)

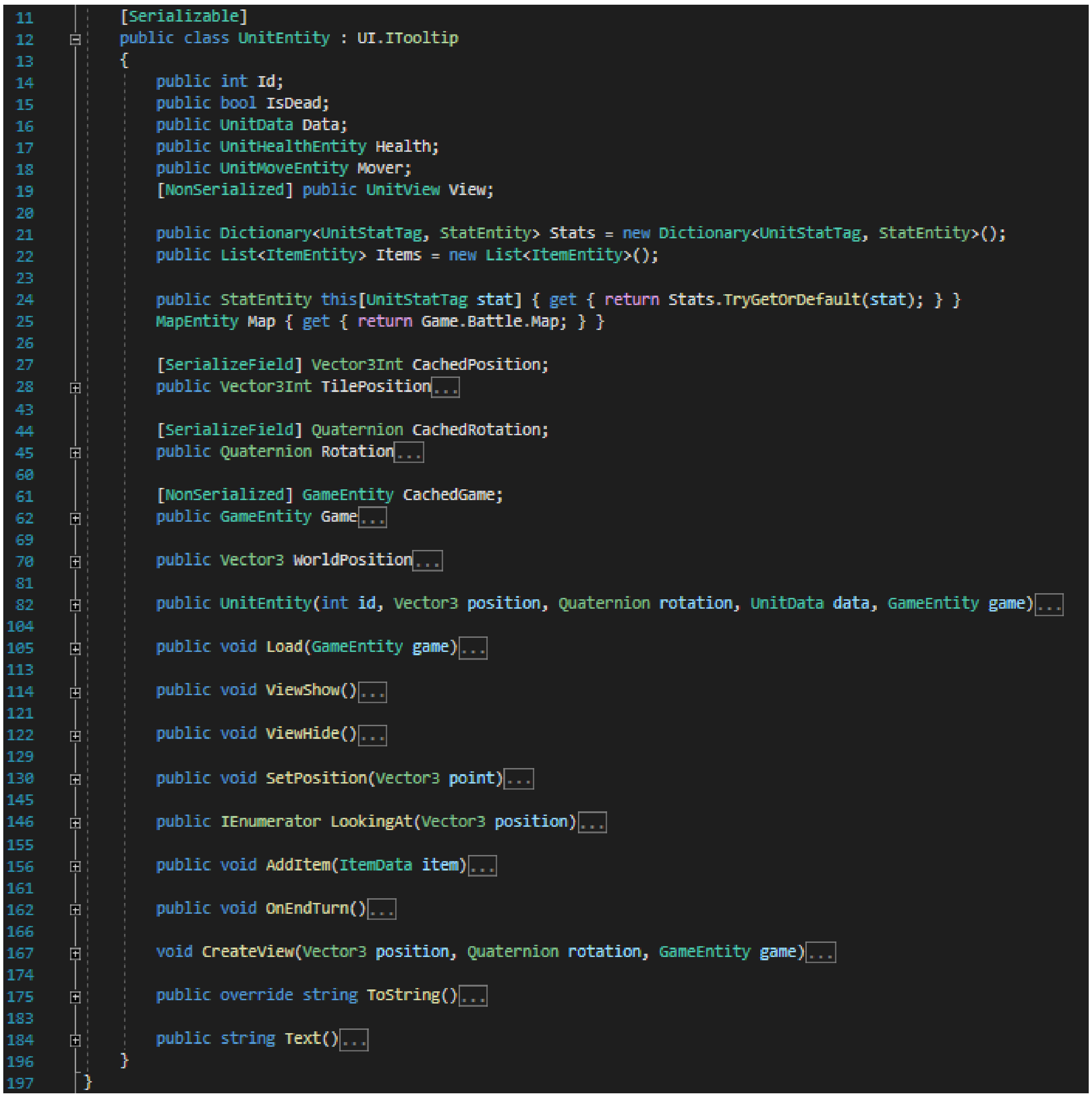
Duplicate | Create

Visual | Stats | DefaultItems

Avatar	UnitRedIcon
Model	UnitRed
Ui Holder	X 0 Y 2 Z 0

Join our Discord

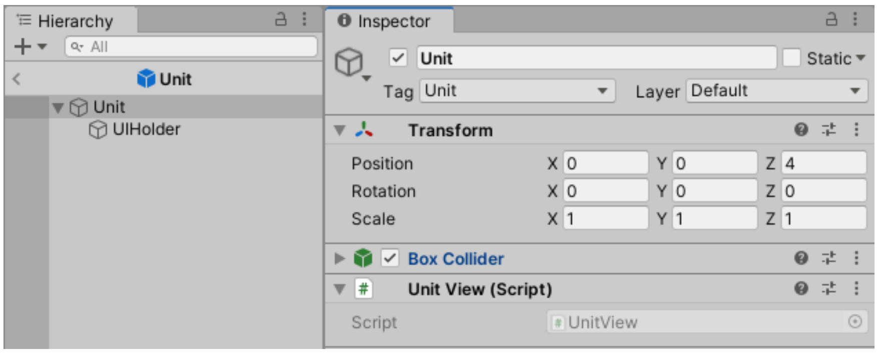
UnitEntity is a wrapper of *UnitData* which contains actual unit state



```
[Serializable]
public class UnitEntity : UI.ITooltip
{
    public int Id;
    public bool IsDead;
    public UnitData Data;
    public UnitHealthEntity Health;
    public UnitMoveEntity Mover;
    [NonSerialized] public UnitView View;

    public Dictionary<UnitStatTag, StatEntity> Stats = new Dictionary<UnitStatTag, StatEntity>();
    public List<ItemEntity> Items = new List<ItemEntity>();

    public StatEntity this[UnitStatTag stat] { get { return Stats.TryGetOrDefault(stat); } }
    MapEntity Map { get { return Game.Battle.Map; } }

    [SerializeField] Vector3Int CachedPosition;
    public Vector3Int TilePosition...

    [SerializeField] Quaternion CachedRotation;
    public Quaternion Rotation...

    [NonSerialized] GameEntity CachedGame;
    public GameEntity Game...

    public Vector3 WorldPosition...

    public UnitEntity(int id, Vector3 position, Quaternion rotation, UnitData data, GameEntity game)...

    public void Load(GameEntity game)...

    public void ViewShow()...

    public void ViewHide()...

    public void SetPosition(Vector3 point)...

    public IEnumerator LookingAt(Vector3 position)...

    public void AddItem(ItemData item)...

    public void OnEndTurn()...

    void CreateView(Vector3 position, Quaternion rotation, GameEntity game)...

    public override string ToString()...

    public string Text()...
    }
}
```

For example, *UnitEntity* contains information is it alive or dead and what kind of items does it have (maybe it was given something new in comparison with default items)

UnitView it a visual representation of *UnitEntity*. It is located inside Unit prefab as a MonoBehaviour component.



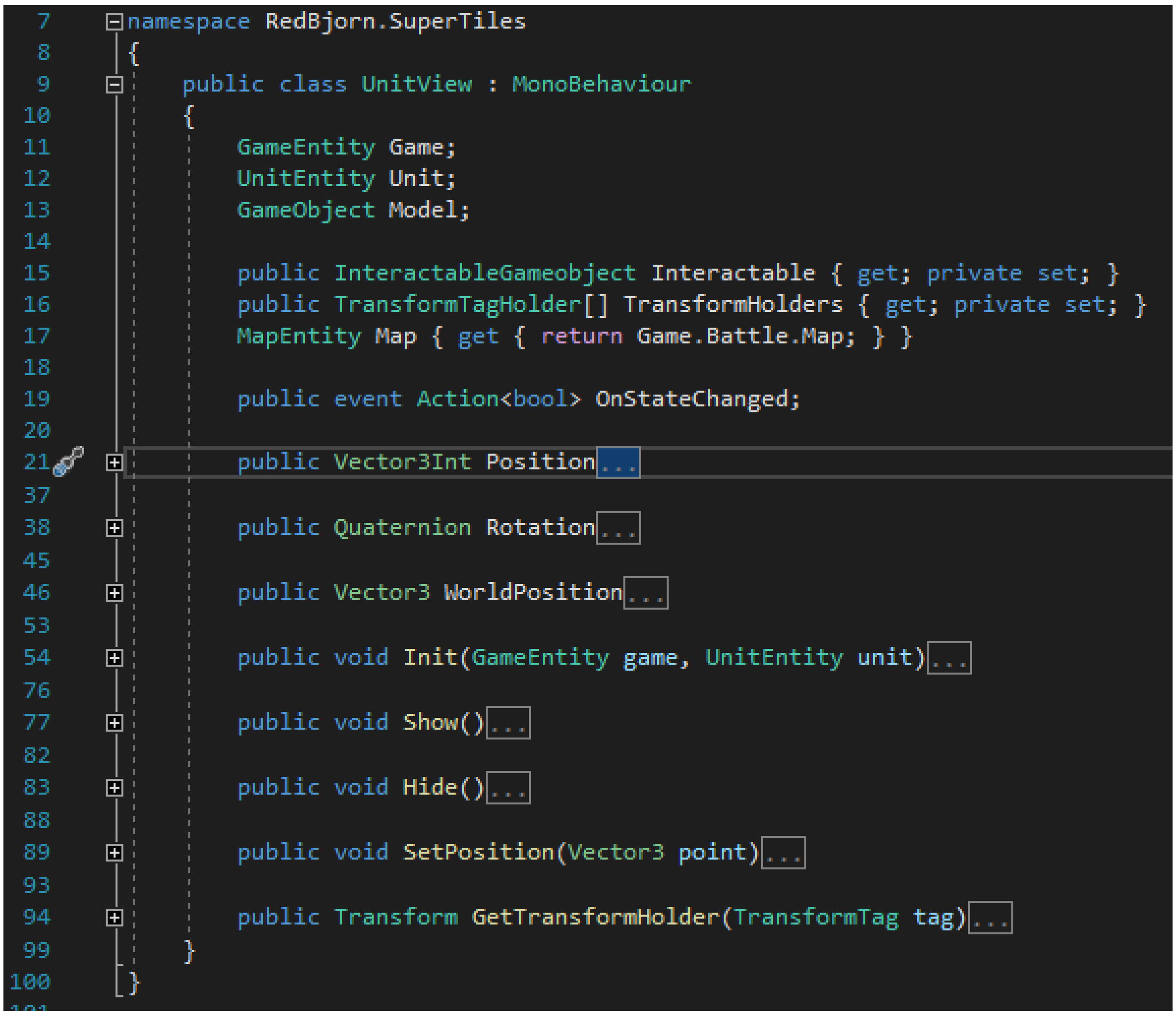
Unit
Tag: Unit   Layer: Default

	X	Y	Z
Position	0	0	4
Rotation	0	0	0
Scale	1	1	1

Script: UnitView

Join our Discord

UnitView view contains information about model that was instantiated for Unit and its world space position.

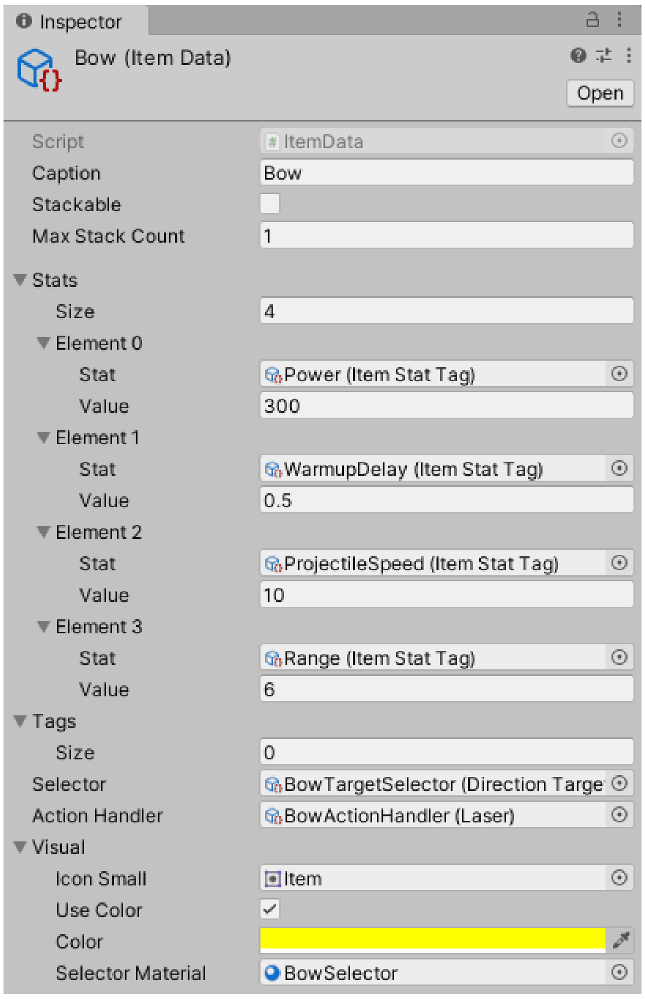


```
namespace RedBjorn.SuperTiles
{
    public class UnitView : MonoBehaviour
    {
        GameEntity Game;
        UnitEntity Unit;
        GameObject Model;

        public InteractableGameobject Interactable { get; private set; }
        public TransformTagHolder[] TransformHolders { get; private set; }
        MapEntity Map { get { return Game.Battle.Map; } }

        public event Action<bool> OnStateChanged;

        public Vector3Int Position...

        public Quaternion Rotation...

        public Vector3 WorldPosition...

        public void Init(GameEntity game, UnitEntity unit)...

        public void Show()...

        public void Hide()...

        public void SetPosition(Vector3 point)...

        public Transform GetTransformHolder(TransformTag tag)...
    }
}
```

Join our Discord

Item

Item in *SuperTiles* consist of 3 main components: *ItemData*, *ItemEntity* and *ItemAction*.

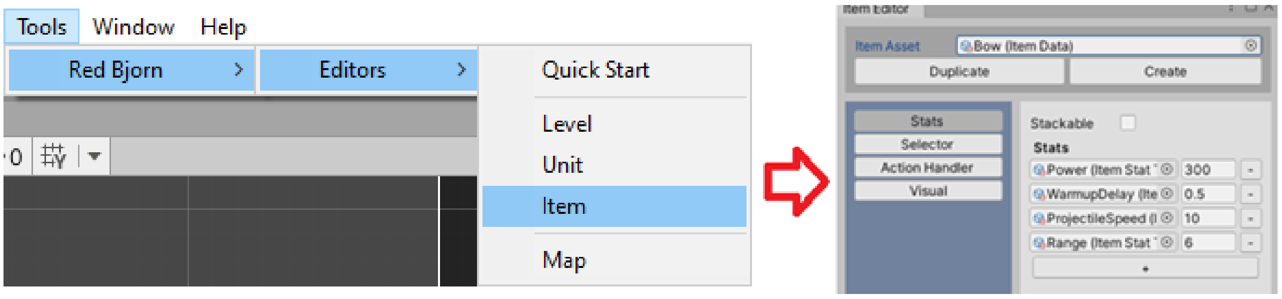


Script	ItemData
Caption	Bow
Stackable	☐
Max Stack Count	1
Stats	
Size	4
Element 0	
Stat	Power (Item Stat Tag)
Value	300
Element 1	
Stat	WarmupDelay (Item Stat Tag)
Value	0.5
Element 2	
Stat	ProjectileSpeed (Item Stat Tag)
Value	10
Element 3	
Stat	Range (Item Stat Tag)
Value	6
Tags	
Size	0
Selector	BowTargetSelector (Direction Targe
Action Handler	BowActionHandler (Laser)
Visual	
Icon Small	Item
Use Color	✓
Color	🟨
Selector Material	BowSelector

ItemData is an information storage with default values like Maximum Cast Range or Power. Besides Stats information, *ItemData* contains visual data (Caption, Icon, Color, and Selector material) and rules:

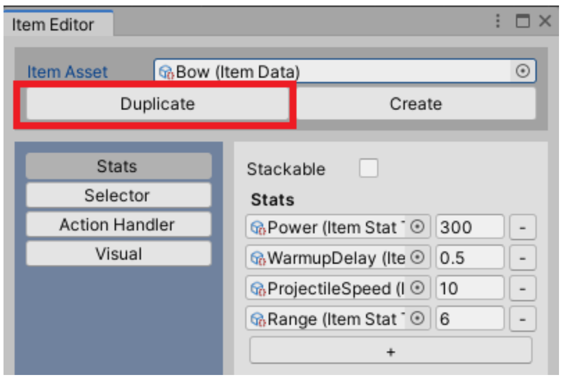
- Tags – item markers for different logic (Health Rules)
- Selector field – how to select valid values to create *ItemAction*
- ActionHandler field– the way *ItemAction* should be played

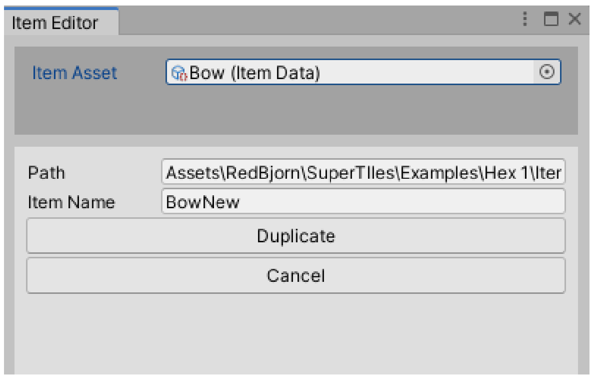
ItemData consist of nested ScriptableObjects and Item Editor window was designed to simplify edit process.



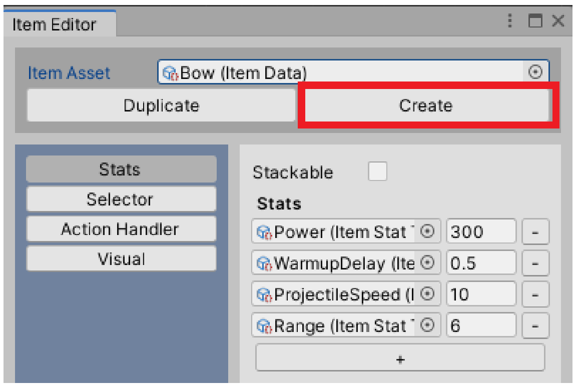
Join our Discord

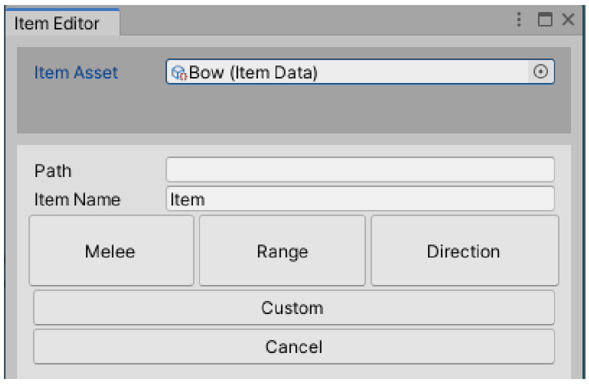
Here you can duplicate selected Item



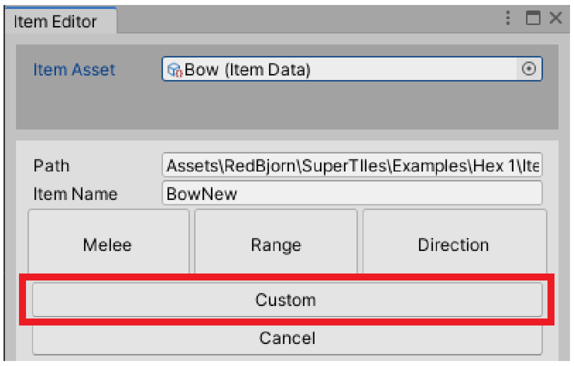


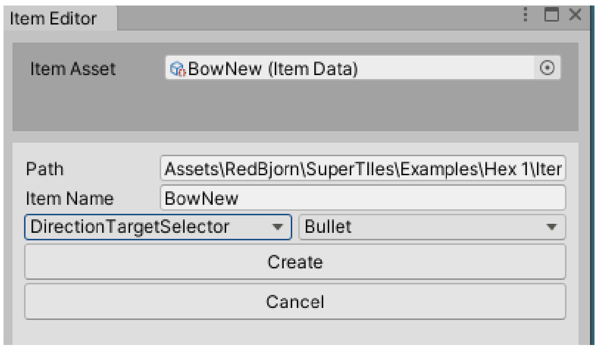
Or create a new one from common templates: Melee, Range Direction





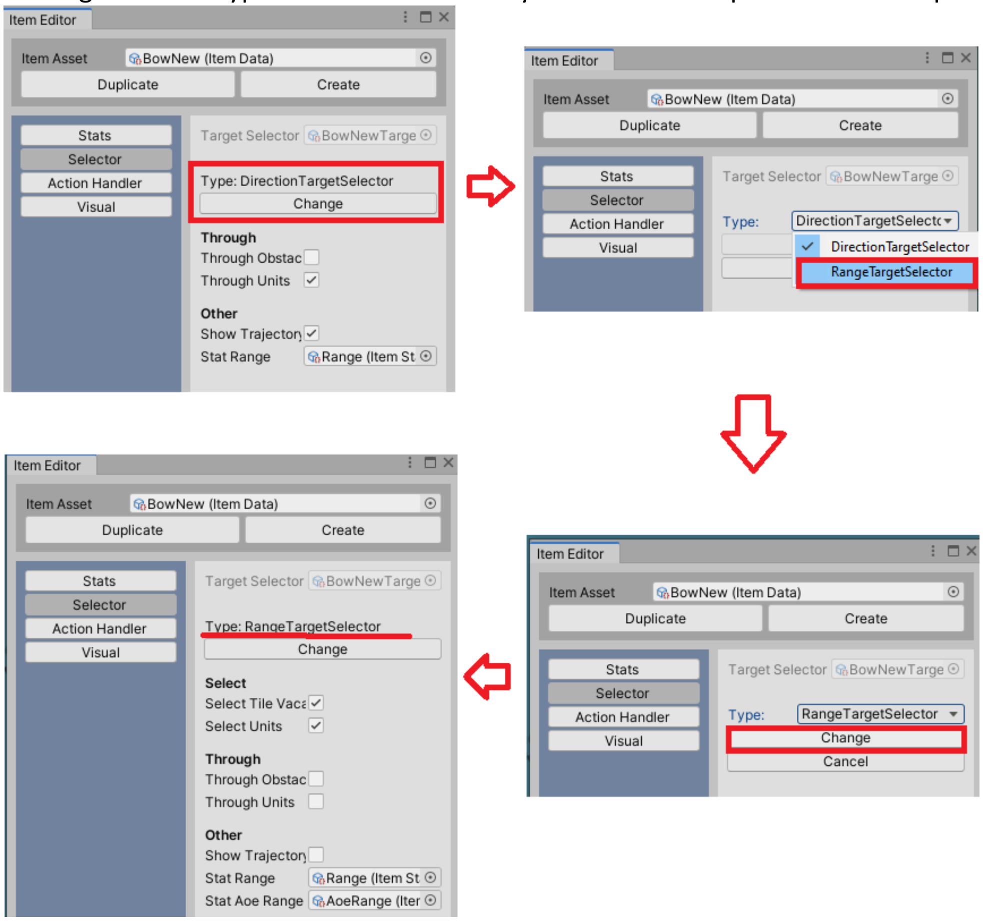
Or you can create an item in manual manner by clicking Custom field and specifying Selector and ActionHandler types



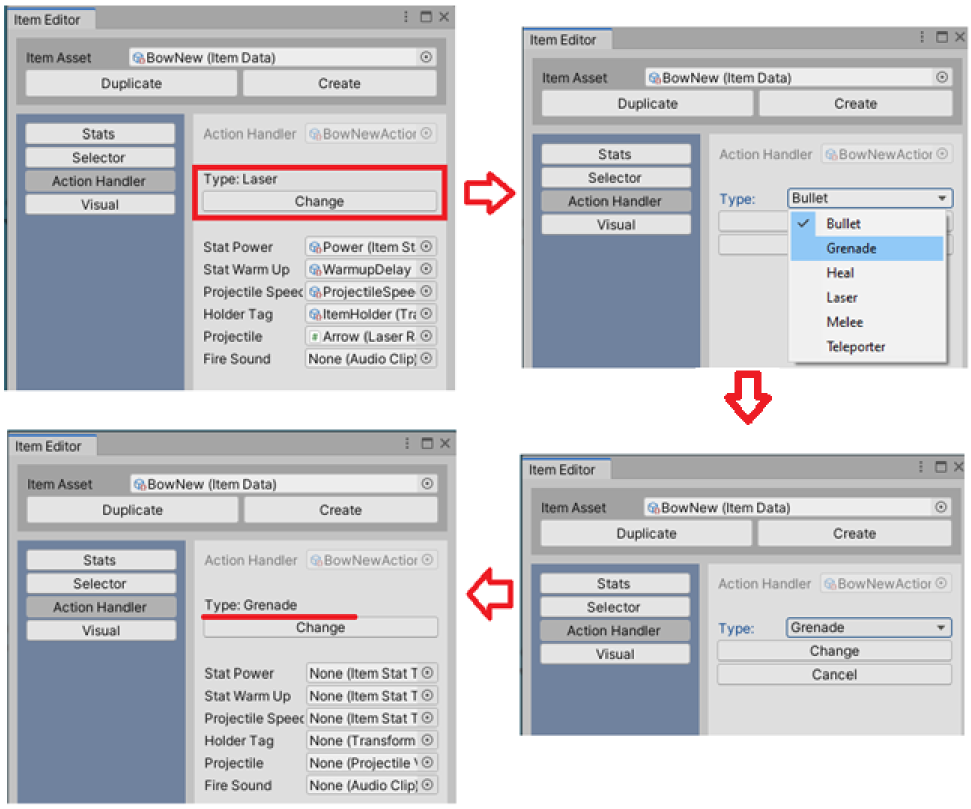


Join our Discord

If you need to change Selector type of existed ItemData you should do steps shown on the picture below

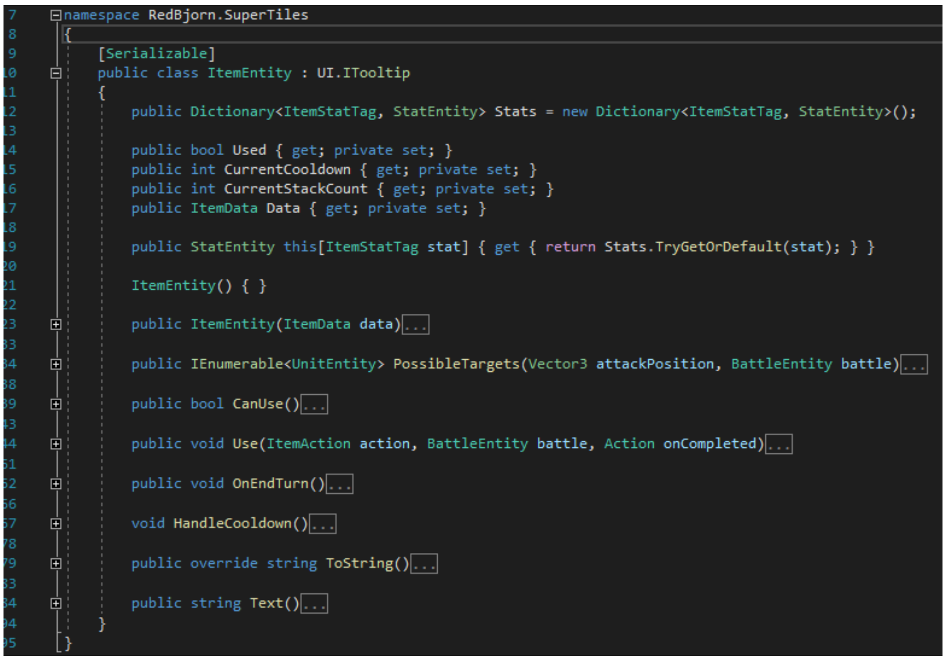


Also, if you need to change Action handler type you should do steps shown on the picture below



[Join our Discord](#)

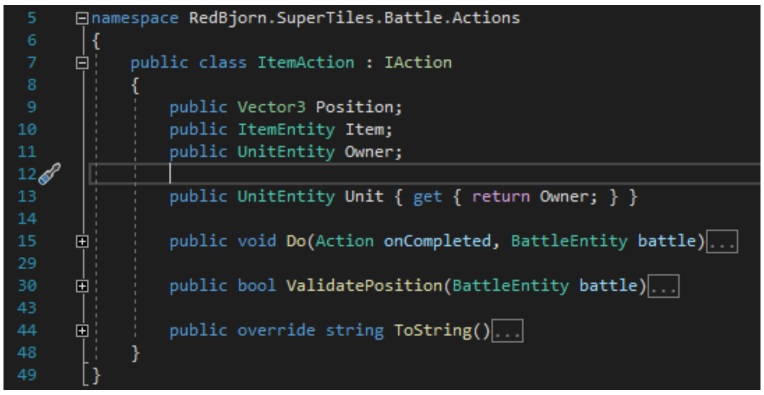
ItemEntity is a wrapper for *ItemData* which contains its current state (CurrentCooldown, CurrentStackCount and etc) and handles data usage.



```
namespace RedBjorn.SuperTiles
{
    [Serializable]
    public class ItemEntity : UI.ITooltip
    {
        public Dictionary<ItemStatTag, StatEntity> Stats = new Dictionary<ItemStatTag, StatEntity>();

        public bool Used { get; private set; }
        public int CurrentCooldown { get; private set; }
        public int CurrentStackCount { get; private set; }
        public ItemData Data { get; private set; }

        public StatEntity this[ItemStatTag stat] { get { return Stats.TryGetOrDefault(stat); } }

        ItemEntity() { }

        public ItemEntity(ItemData data)...

        public IEnumerable<UnitEntity> PossibleTargets(Vector3 attackPosition, BattleEntity battle)...

        public bool CanUse()...

        public void Use(ItemAction action, BattleEntity battle, Action onCompleted)...

        public void OnEndTurn()...

        void HandleCooldown()...

        public override string ToString()...

        public string Text()...
    }
}
```

ItemAction is the result of the *ItemData -> Selector* work. It is easy to read *ItemAction* as

Unit use Item at the Position.

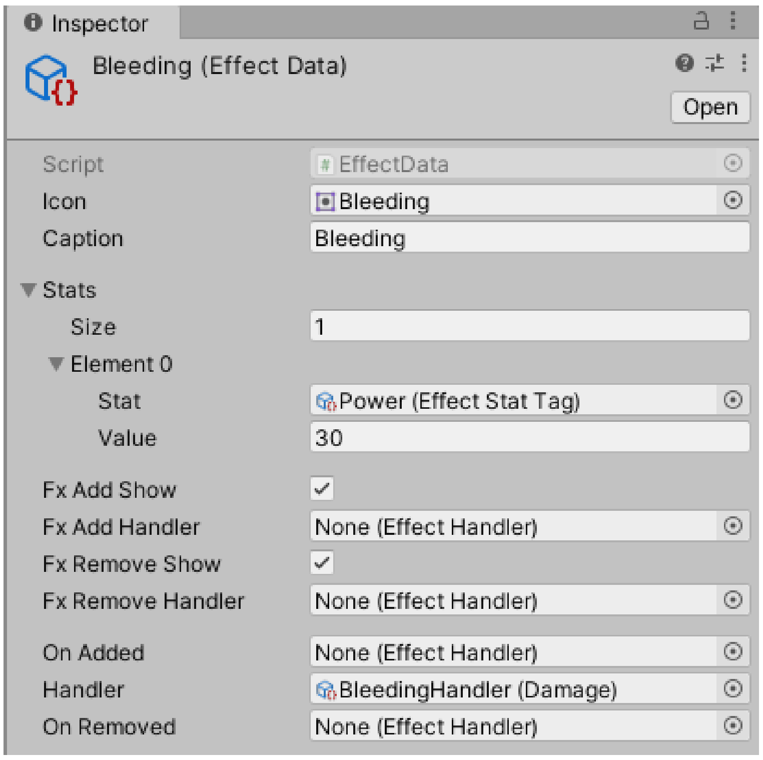


```
namespace RedBjorn.SuperTiles.Battle.Actions
{
    public class ItemAction : IAction
    {
        public Vector3 Position;
        public ItemEntity Item;
        public UnitEntity Owner;

        public UnitEntity Unit { get { return Owner; } }

        public void Do(Action onCompleted, BattleEntity battle)...

        public bool ValidatePosition(BattleEntity battle)...

        public override string ToString()...
    }
}
```

Join our Discord

Effect

Effect in *SuperTiles* consist of 2 main components: *EffectData*, *EffectEntity*.



Inspector

Bleeding (Effect Data)

Open

Script	EffectData
Icon	Bleeding
Caption	Bleeding
Stats	
Size	1
Element 0	
Stat	Power (Effect Stat Tag)
Value	30
Fx Add Show	✓
Fx Add Handler	None (Effect Handler)
Fx Remove Show	✓
Fx Remove Handler	None (Effect Handler)
On Added	None (Effect Handler)
Handler	BleedingHandler (Damage)
On Removed	None (Effect Handler)

First one is *EffectData* which is is an information storage with Stats values, Visual info, and the logic. Logic consists of 3 parts:

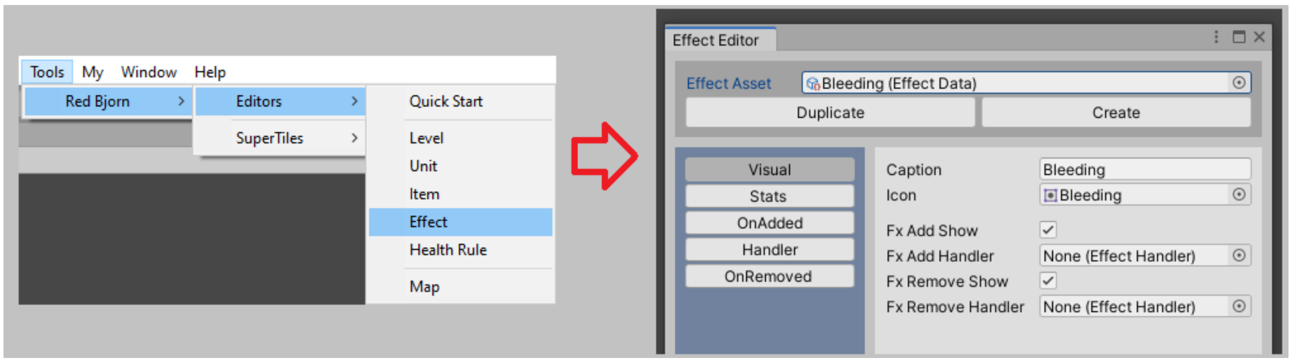
- **OnAdded** executes when effect was added to Unit
- **Handler** executes every owner's turn, before all actions
- **OnRemoved** executes when effect was removed from Unit.

For example, **Bleeding** effect at the screenshot above haven't got **OnAdded** and **OnRemoved** logic. Consequently, effect only deals damage (**Handler** has **Damage** type).

Also, you could play Fx when effect will be added or removed by checking corresponding toggles. One moment:

If **Fx Add (Remove) Handler** is **None** – Default Fx (located inside asset **EffectFxAddDefault** (**EffectFxRemoveDefault**)) will be played

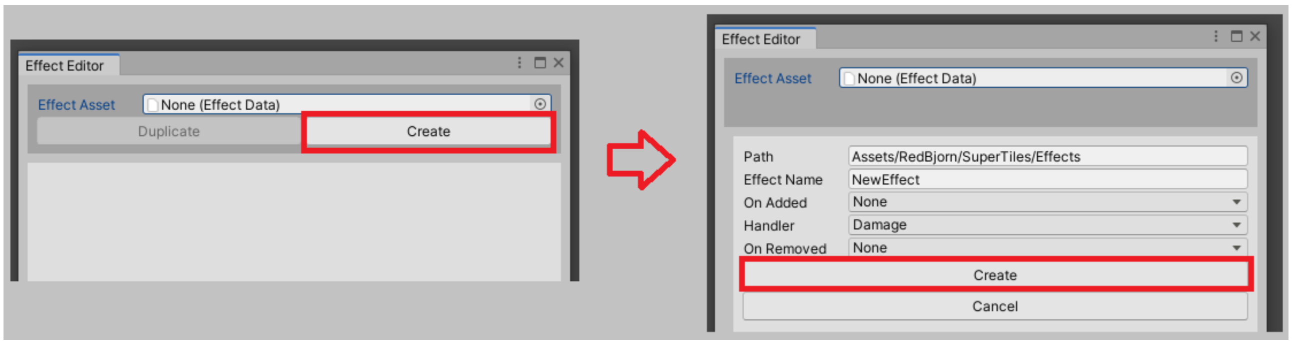
EffectData consists of nested ScriptableObjects and Effect Editor window was designed to simplify edit process



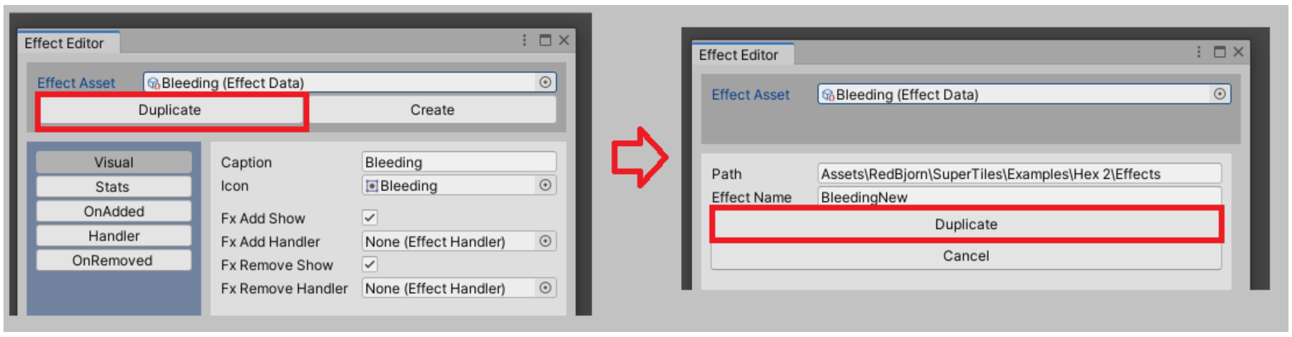
Caption	Bleeding
Icon	Bleeding
Fx Add Show	✓
Fx Add Handler	None (Effect Handler)
Fx Remove Show	✓
Fx Remove Handler	None (Effect Handler)

Join our Discord

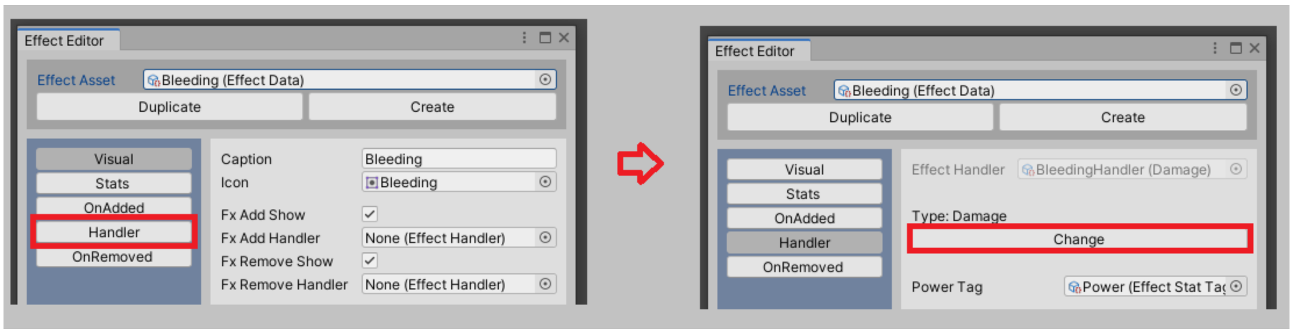
Here you can create a new one



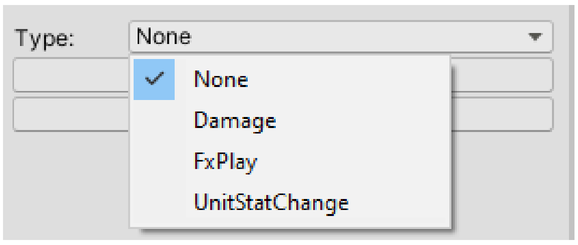
Or duplicate selected *EffectData*



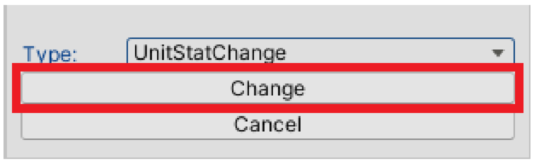
If you decide to change any logic (OnAdded, Handler, OnRemoved), just select corresponding tab and Click Change button



Select new type from popup. If you select **None** than existing logic will be deleted.



And click Change to submit you decision



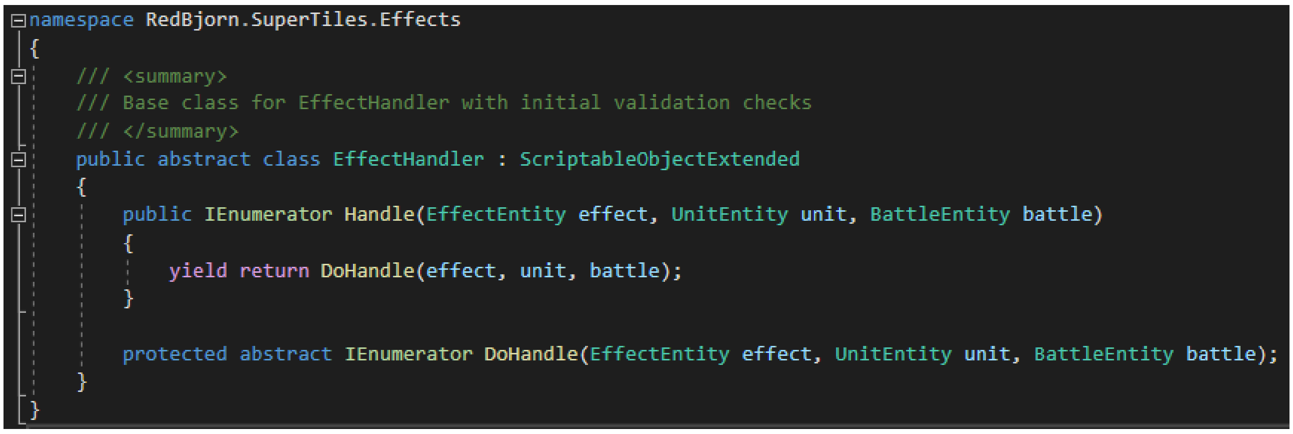
Join our Discord

If you create Effect without any logic, Effect transforms into StatusEffect. It does nothing but could be a marker for any other logic. By other logic, we mean Health Rules

There are 3 types of Handler logic:

1. Damage – deal damage every unit turn. Power is defined as a stat in Stats list
2. FxPlay – Spawn object for specified duration and then destroy it
3. UnitStatChange – change specified unit stat

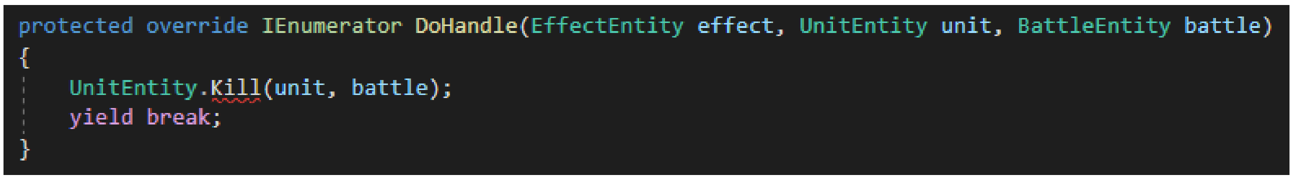
List of Handlers will be extended in future updates, but you could easily implement any logic by yourself. For example, you decide to create effect that will kill unit when will be removed. Of course, you could create effect that deal -9999999 damage, but sometimes this way could lead to unexpected results (input damage could be converted to a small value and the unit will survive). Let's assume that we still decide to create own logic, then we should create a new class **Kill** which should inherit **EffectHandler** class.



```
namespace RedBjorn.SuperTiles.Effects
{
    /// <summary>
    /// Base class for EffectHandler with initial validation checks
    /// </summary>
    public abstract class EffectHandler : ScriptableObjectExtended
    {
        public IEnumerator Handle(EffectEntity effect, UnitEntity unit, BattleEntity battle)
        {
            yield return DoHandle(effect, unit, battle);
        }

        protected abstract IEnumerator DoHandle(EffectEntity effect, UnitEntity unit, BattleEntity battle);
    }
}
```

As you could see, EffectHandler has only one abstract method which just needs to be implemented.

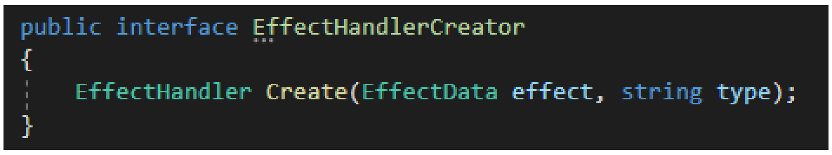
Something like



```
protected override IEnumerator DoHandle(EffectEntity effect, UnitEntity unit, BattleEntity battle)
{
    UnitEntity.Kill(unit, battle);
    yield break;
}
```

Method **Kill** is underlined because there is no such method, so I wrote it just for example. There is no doubt that you could implement it by yourself.

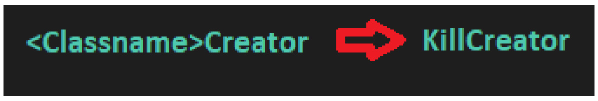
And one more step, you should create **Kill** class creator. To do this you need to create a new class which will implement **EffectHandlerCreator** interface. It is needed for correct Effect Editor work.



```
public interface EffectHandlerCreator
{
    EffectHandler Create(EffectData effect, string type);
}
```

Examples could be found inside **EffectHandlerCreator** file.

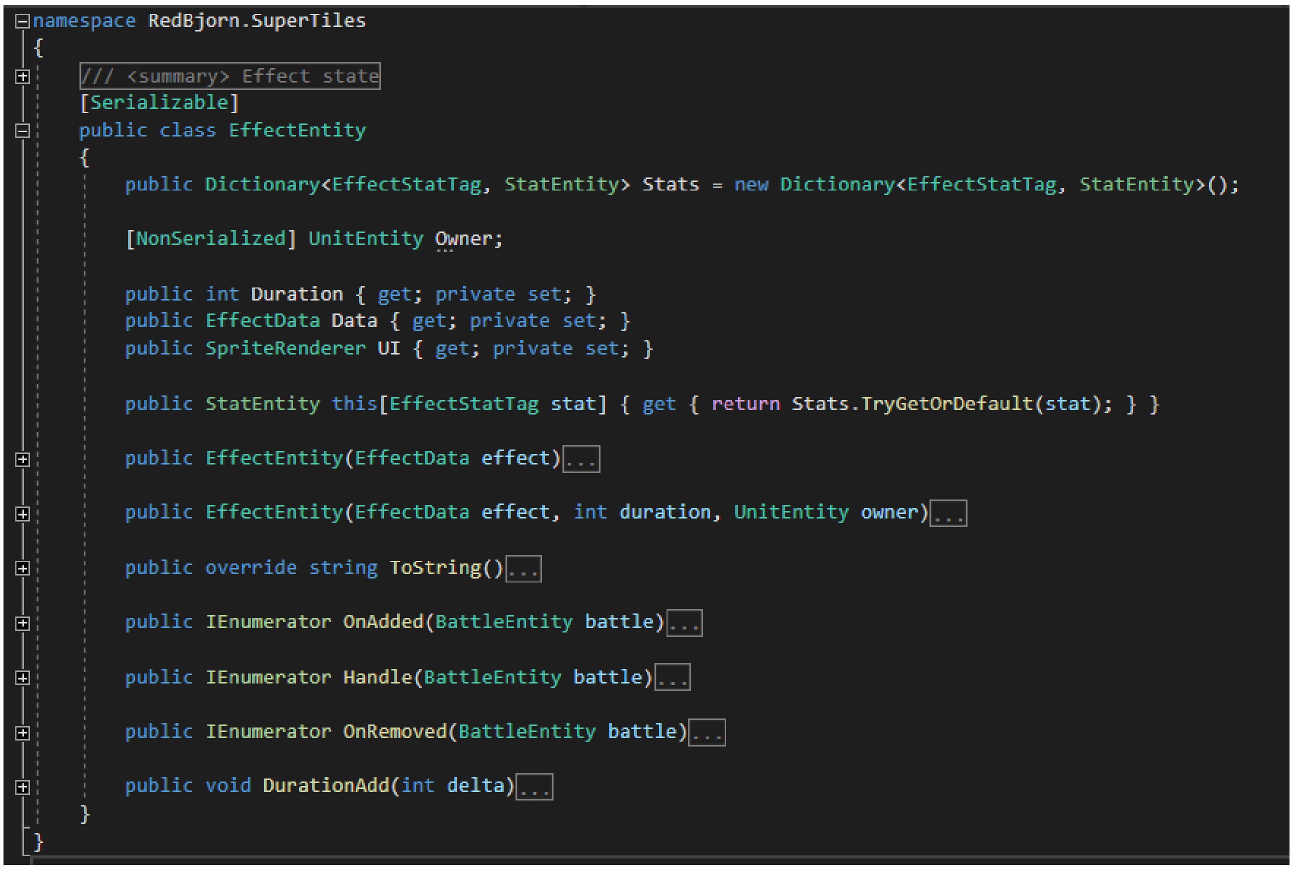
Important note: Creator class name should be made in the following form:



<Classname>Creator ⇨ KillCreator

Join our Discord

Second important Effect component is *EffectEntity*.

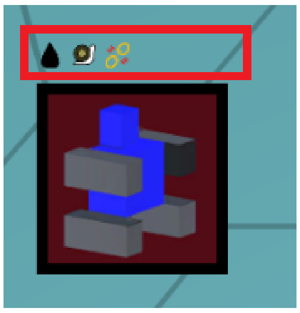


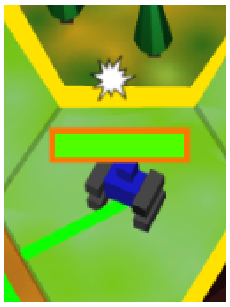
```
namespace RedBjorn.SuperTiles
{
    /// <summary> Effect state
    [Serializable]
    public class EffectEntity
    {
        public Dictionary<EffectStatTag, StatEntity> Stats = new Dictionary<EffectStatTag, StatEntity>();

        [NonSerialized] UnitEntity Owner;

        public int Duration { get; private set; }
        public EffectData Data { get; private set; }
        public SpriteRenderer UI { get; private set; }

        public StatEntity this[EffectStatTag stat] { get { return Stats.TryGetOrDefault(stat); } }

        public EffectEntity(EffectData effect)...

        public EffectEntity(EffectData effect, int duration, UnitEntity owner)...

        public override string ToString()...

        public IEnumerator OnAdded(BattleEntity battle)...

        public IEnumerator Handle(BattleEntity battle)...

        public IEnumerator OnRemoved(BattleEntity battle)...

        public void DurationAdd(int delta)...
    }
}
```

Here we store Effect state at the runtime. Effect state mainly consist of effect current duration and several wrappers of *EffectData* logic methods.

Usually, Effects could be added from Item -> ActionHandler. More info here.

Effects will be visualized near unit avatar and there will be an effect mark near unit model.

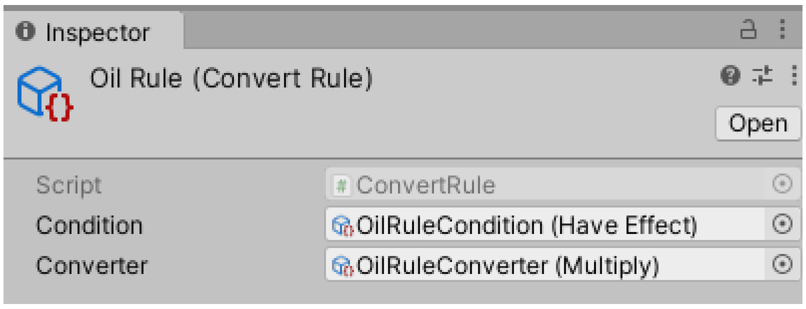




Join our Discord

Health Rules

It is a storage for a list of rules. Each rule is a single ScriptableObject of *CovertRule* type.



Oil Rule (Convert Rule)

Script: ConvertRule
Condition: OilRuleCondition (Have Effect)
Converter: OilRuleConverter (Multiply)

CovertRule consist of two fields: Condition and Converter. It could be read as If **Condition** is true than convert input delta value with **Converter**.

Condition field has a **Condition** class type which is an abstract class with one method.

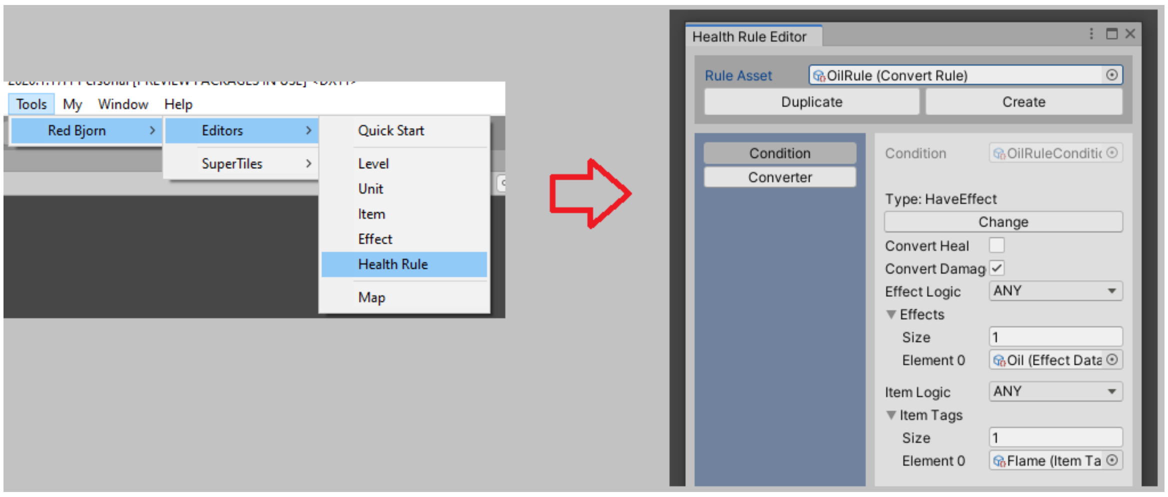
```
namespace RedBjorn.SuperTiles.Health
{
    public abstract class Condition : ScriptableObjectExtended
    {
        public abstract bool IsMet(float delta, UnitEntity victim, UnitEntity damager, ItemEntity item);
    }
}
```

You can see that condition could be checked with any of victim, damager and item or all of them at once.

Converter filed has a **ValueConverter** class type which is also an abstract class with one method.

```
namespace RedBjorn.SuperTiles.Health
{
    public abstract class ValueConverter : ScriptableObjectExtended
    {
        public abstract float Convert(float val);
    }
}
```

Health Rule Editor window was designed to simplify the process of editing single rule.

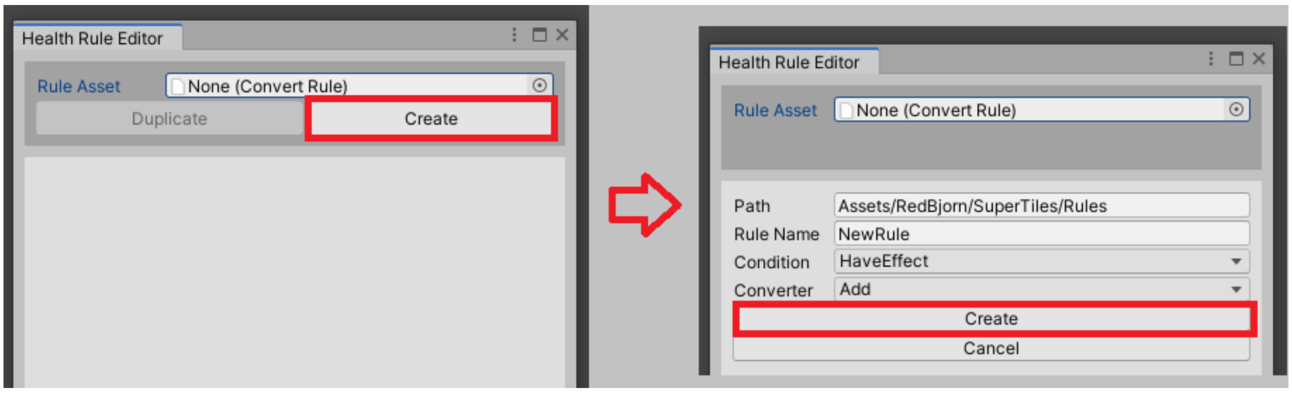


Rule Asset: OilRule (Convert Rule)

Condition: OilRuleConditic
Type: HaveEffect
Convert Heal: ☐
Convert Damag: ✓
Effect Logic: ANY
Effects
Size: 1
Element 0: Oil (Effect Data
Item Logic: ANY
Item Tags
Size: 1
Element 0: Flame (Item Ta

Join our Discord

Here you can create a new one


Health Rule Editor

Rule Asset: None (Convert Rule)

Duplicate | Create

Health Rule Editor

Rule Asset: None (Convert Rule)

Path: Assets/RedBjorn/SuperTiles/Rules

Rule Name: NewRule

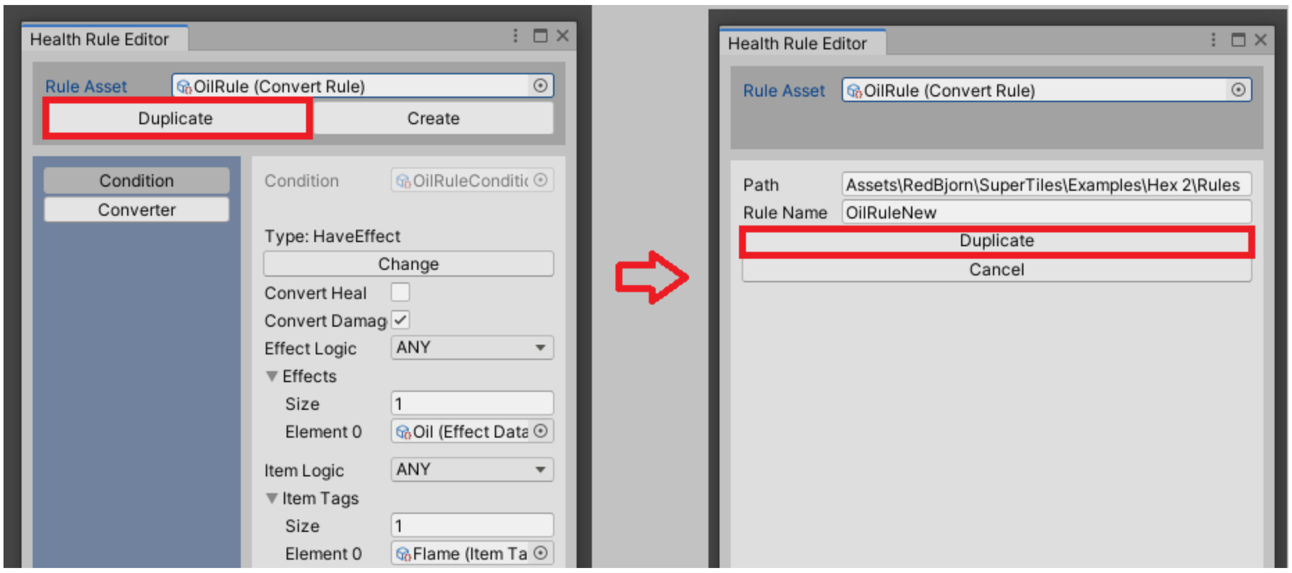
Condition: HaveEffect

Converter: Add

Create

Cancel

Or duplicate selected CovertRule


Health Rule Editor

Rule Asset: OilRule (Convert Rule)

Duplicate | Create

Condition

Converter

Condition: OilRuleConditic

Type: HaveEffect

Change

Convert Heal: ☐

Convert Damag: ✓

Effect Logic: ANY

Effects

Size: 1

Element 0: Oil (Effect Data

Item Logic: ANY

Item Tags

Size: 1

Element 0: Flame (Item Ta

Health Rule Editor

Rule Asset: OilRule (Convert Rule)

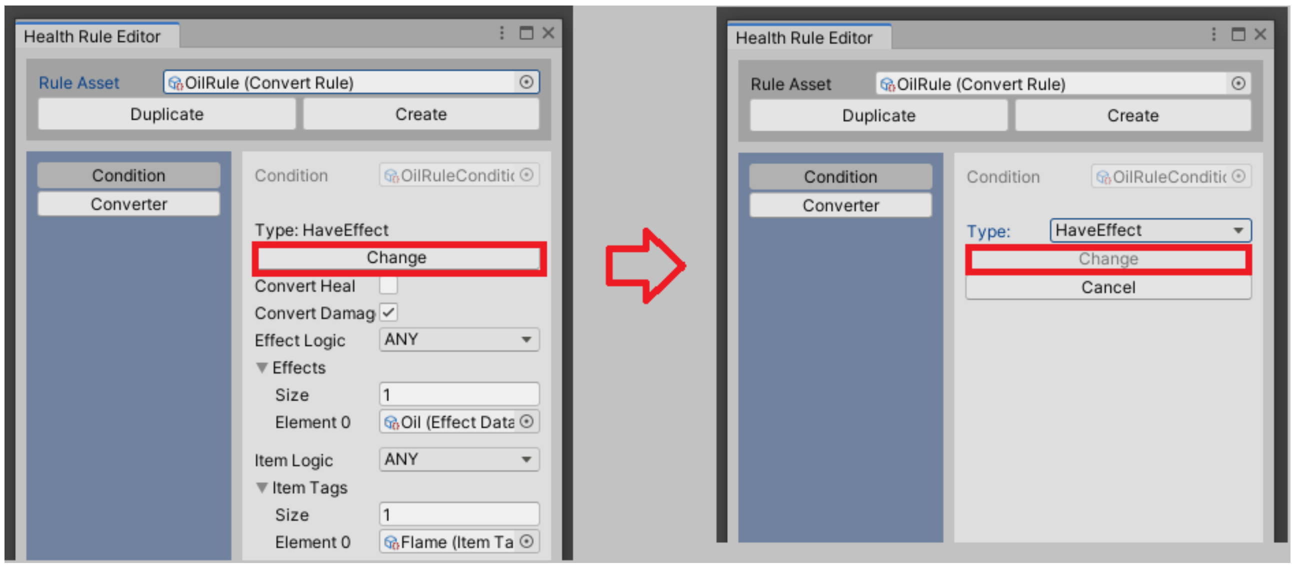
Path: Assets\RedBjorn\SuperTiles\Examples\Hex 2\Rules

Rule Name: OilRuleNew

Duplicate

Cancel

Also, you could change **Condition** type by clicking Change button


Health Rule Editor

Rule Asset: OilRule (Convert Rule)

Duplicate | Create

Condition

Converter

Condition: OilRuleConditic

Type: HaveEffect

Change

Convert Heal: ☐

Convert Damag: ✓

Effect Logic: ANY

Effects

Size: 1

Element 0: Oil (Effect Data

Item Logic: ANY

Item Tags

Size: 1

Element 0: Flame (Item Ta

Health Rule Editor

Rule Asset: OilRule (Convert Rule)

Duplicate | Create

Condition

Converter

Condition: OilRuleConditic

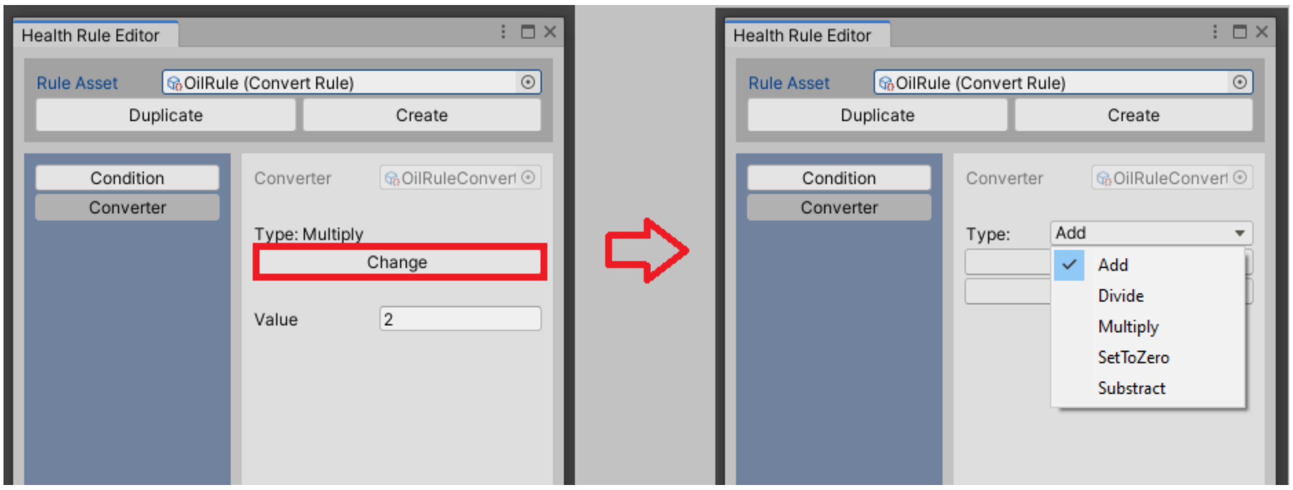
Type: HaveEffect

Change

Cancel

Join our Discord

And you could change **Converter** type by clicking Change button



To implement your own **Conditions** and **Converters** you should create corresponding Creator classes besides:

```
public interface ConditionCreator
{
    Condition Create(ConvertRule rule, string type);
}
```

```
public interface ValueConverterCreator
{
    ValueConverter Create(ConvertRule rule, string type);
}
```

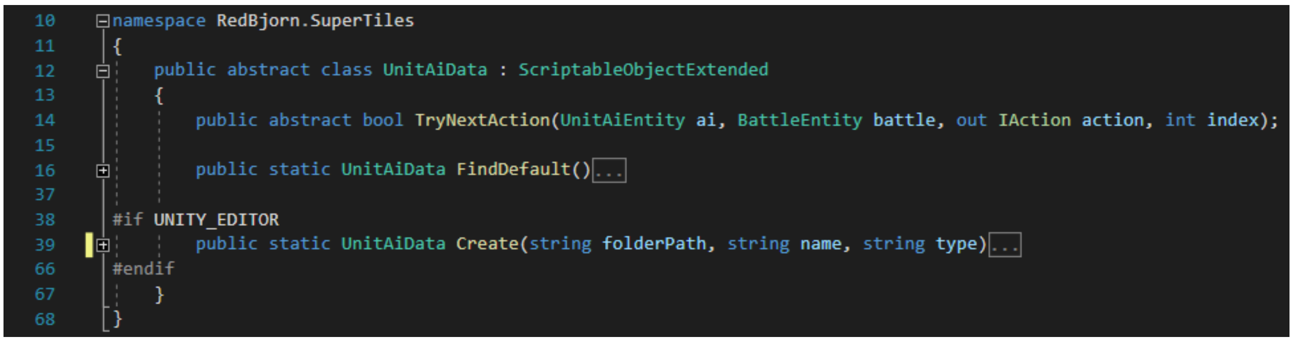
It is similar to EffectHandler creation. More info here

Join our Discord

AI

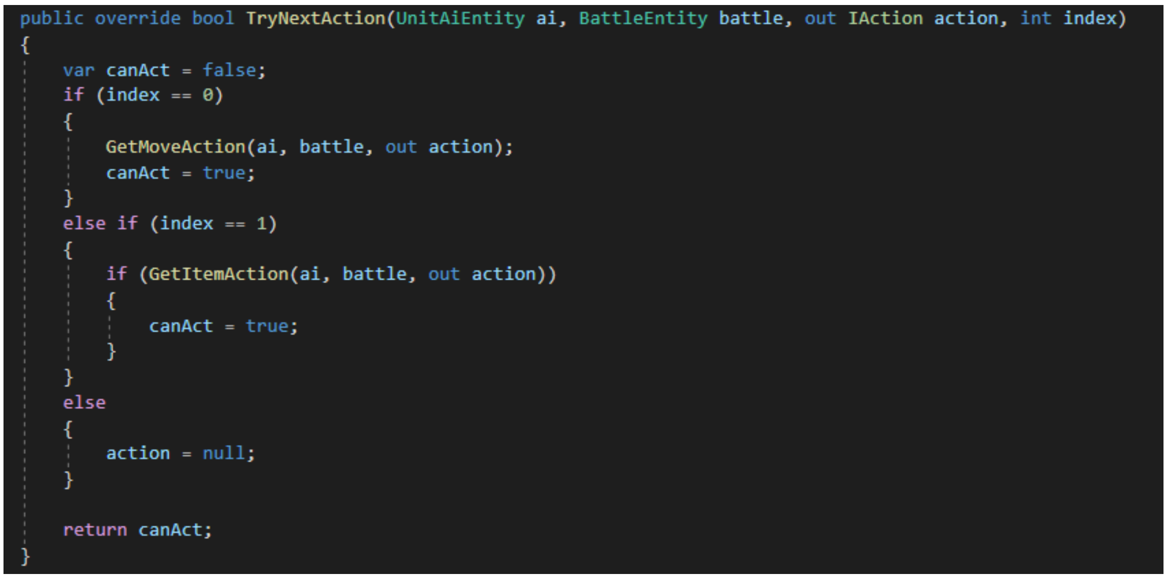
Ai in *SuperTiles* consist of 2 main components: *UnitAiData* and *UnitAiEntity*

UnitAiData represent abstract class which successors represent Ai logic by implementing `TryNextAction` method



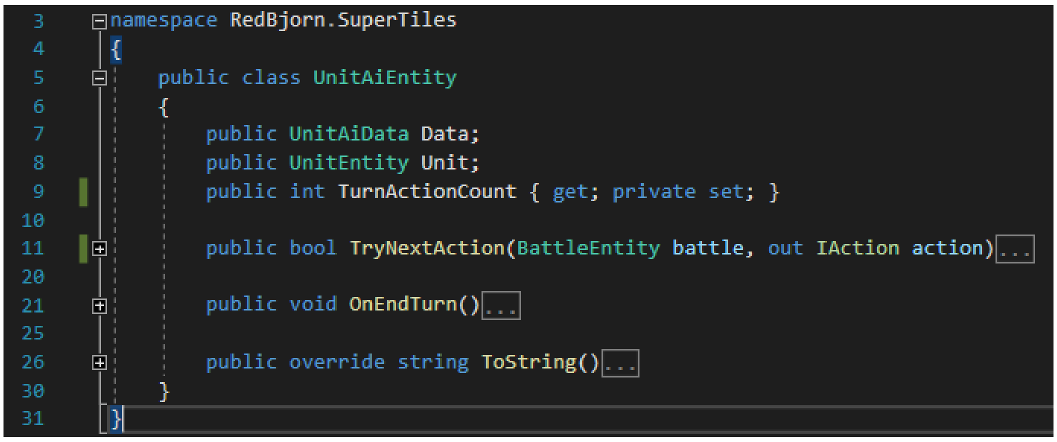
```
namespace RedBjorn.SuperTiles
{
    public abstract class UnitAiData : ScriptableObjectExtended
    {
        public abstract bool TryNextAction(UnitAiEntity ai, BattleEntity battle, out IAction action, int index);

        public static UnitAiData FindDefault()...

#if UNITY_EDITOR
        public static UnitAiData Create(string folderPath, string name, string type)...
#endif
    }
}
```

Current version of *SuperTiles* contains only one Ai implementation: Simple (Simple.cs). Inside this logic Unit try to do 1 move action and then 1 item action



```
public override bool TryNextAction(UnitAiEntity ai, BattleEntity battle, out IAction action, int index)
{
    var canAct = false;
    if (index == 0)
    {
        GetMoveAction(ai, battle, out action);
        canAct = true;
    }
    else if (index == 1)
    {
        if (GetItemAction(ai, battle, out action))
        {
            canAct = true;
        }
    }
    else
    {
        action = null;
    }

    return canAct;
}
```

UnitAiEntity is a wrapper of *UnitAiData* containing runtime state of Ai such as unit it controls, and actions count which were already scheduled for current Ai.



```
namespace RedBjorn.SuperTiles
{
    public class UnitAiEntity
    {
        public UnitAiData Data;
        public UnitEntity Unit;
        public int TurnActionCount { get; private set; }

        public bool TryNextAction(BattleEntity battle, out IAction action)...

        public void OnEndTurn()...

        public override string ToString()...
    }
}
```

Join our Discord

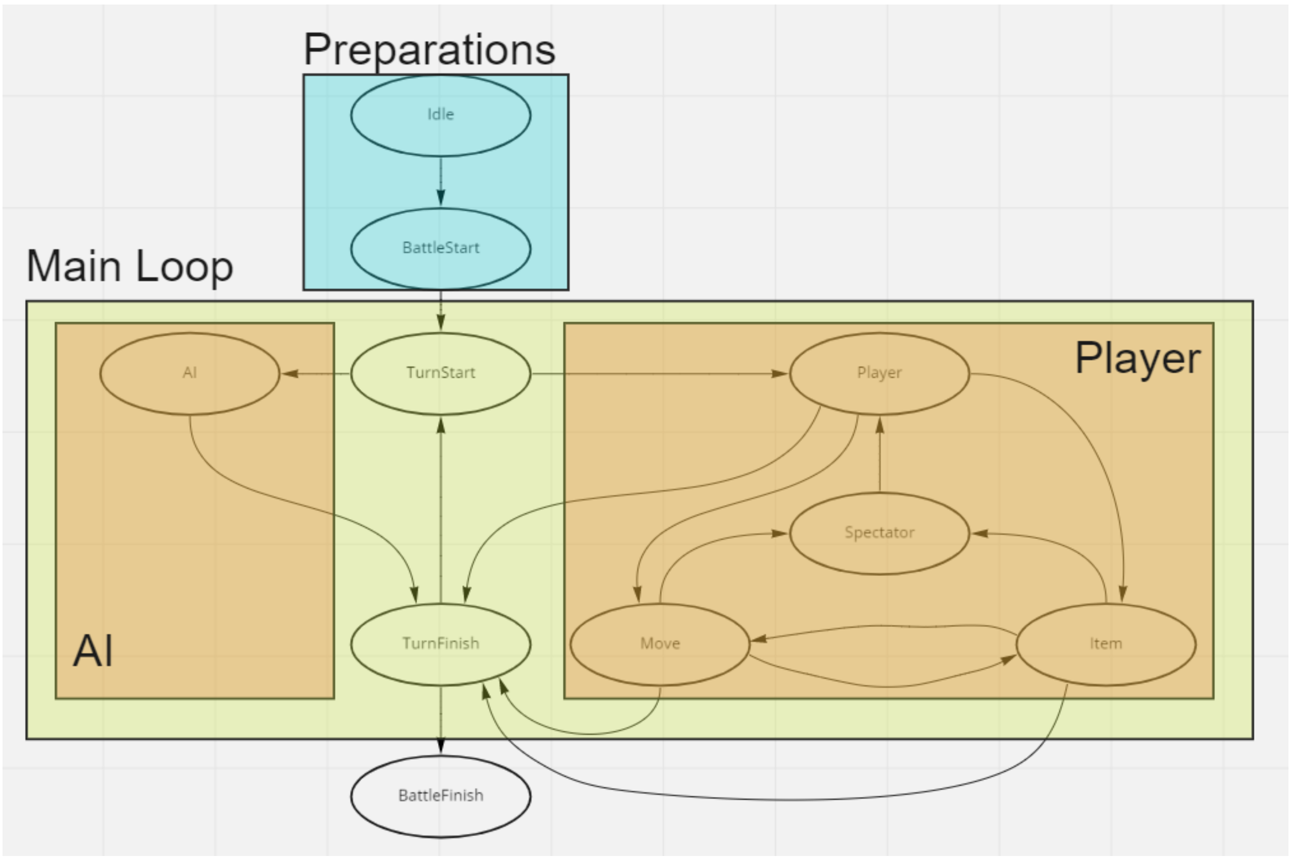
Battle

Battle in *SuperTiles* consist of 3 main components: *BattleSettings*, *BattleEntity* and *BattleView*

Information about *BattleSettings* is here.

BattleView is the main class which defines communication between all entities and visual game representation (View suffix indicates that this class is usually a MonoBehaviour successor and have corresponding entity class, this time it will be *BattleEntity*). This class is an implementation of Finite State Machine (FSM) which control the Battle state.

In the picture below you could see diagram of the FSM.



Preparations

BattleView starts from Idle state where it does nothing. It only waits a signal to start Battle. After this signal is received *BattleView* calculate unit turn order inside BattleStart state and transit to TurnStart state.

Main Loop

Turn state is a place where current Player is defined according to calculated unit turn order at **Preparations** stage. Besides an answer to the question about the type of current player defines what state will be next: **Ai** or **Player**.

If **Ai** state is selected than *AiEntity* calculate all actions that should be done for every controlled unit in consequential manner.

If **Player** state is selected than *PlayerEntity* (this class represent real player) creates move and item actions inside corresponding states Move and Item. When single action is created Player wait the finish of action playing process inside Spectator state. Spectator state is like Idle state but allow actions which do not influence on the game state, like switching between controlled units.

After all actions inside **Ai** or **Player** state are done TurnFinish state takes the control. It updates map state, unit states and item states (cooldown) and check if finish conditions meet.

- If meet, *BattleView* change state to BattleFinish where it displays congratulations window and stops future state transitions.
- If do not meet, *BattleView* starts another cycle by moving to already known TurnStart state

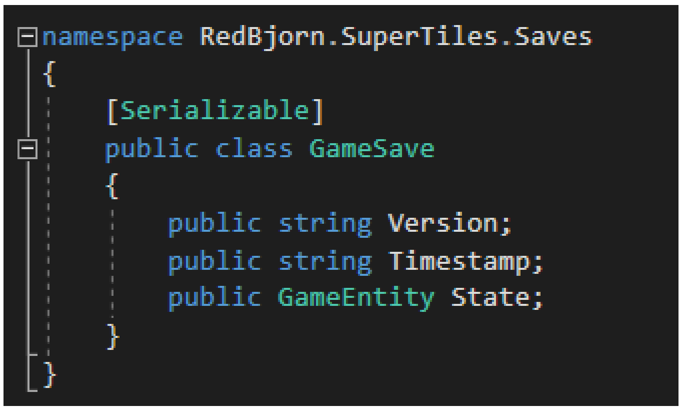
BattleEntity contains current Battle state including:

- Current turn
- Current turn state (Starting, Started, Finishing, Finished)
- Battle state (Started, Finished)
- Players who participate in current Battle
- Dead and alive units
- Ai instances
- and more

Save

Save in *SuperTiles* is organized as a composition on C# plain classes. As Unity doesn't support reference serialization by default, all fields with reference types inherited from *System.Object* should be marked with SerializeReference attribute.

Class which contains whole game info is called *GameSave*.



```csharp
namespace RedBjorn.SuperTiles.Saves
{
    [Serializable]
    public class GameSave
    {
        public string Version;
        public string Timestamp;
        public GameEntity State;
    }
}
```

It consists only of 3 fields:

1. Version field to check compatibility between current build and save file. And if versions mismatch then several handlers could be made to update save file (not implemented it this asset)
2. Timestamp could be used to form appropriate save order.
3. GameEntity stores info about level that should be loaded and the battle state

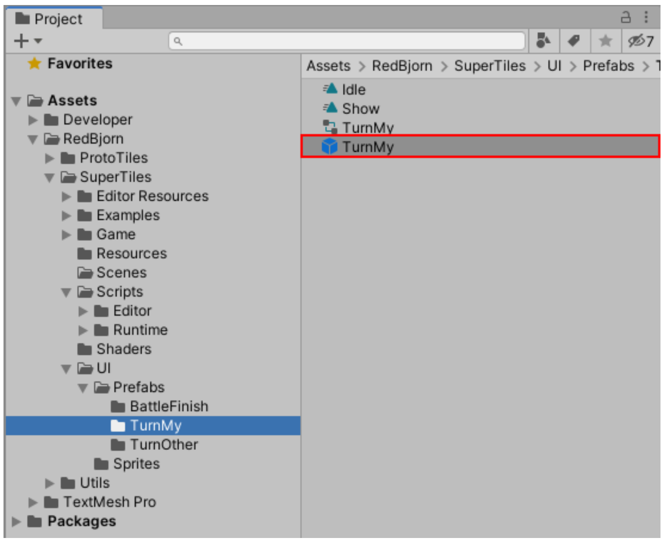
SaveController creates an instance of *GameSave* class inside Save method logic.

Join our Discord

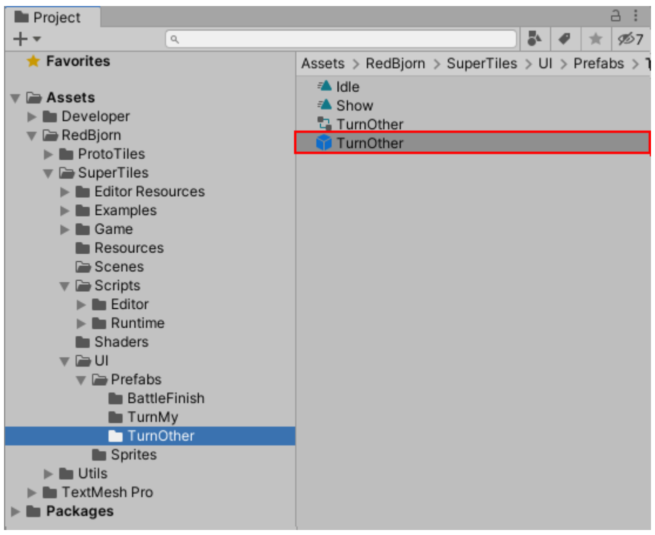
Labels

There are three UI labels which are presented at the beginning of the turn

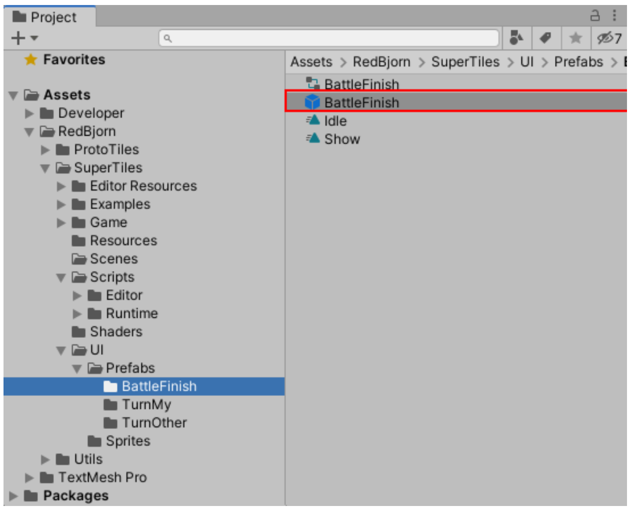
1) **TurnMy** prefab is shown when the player turn starts.



2) **TurnOther** prefab is shown when another player turn starts.



3) **BattleFinish** prefab is shown the battle finishes.



Join our Discord

Controllers

In *SuperTiles*, by Controller we mean global logic dedicated to one side of game which is available from everywhere. It could be also called Manager.

AudioController

Controller which could play any audio clip or already defined button click sound

CameraController

Controller which gets data from InputController and converts it to the movement of Camera

InputController

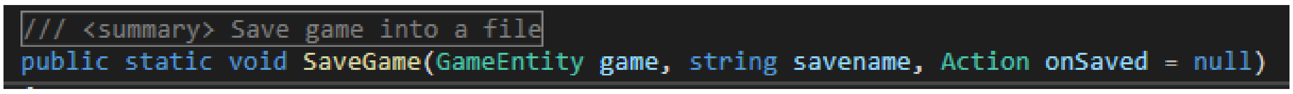
Controller which is a wrapper to UnityEngine.Input class with some quality-of-life features and custom restrictions.

SaveController

Controller which implements basic Save/Load/Delete logic for the specified slot. By slot we mean unique string value.

Save

To save the game, you should pass an instance of *GameEntity* to this controller and a string which represents a slot name.



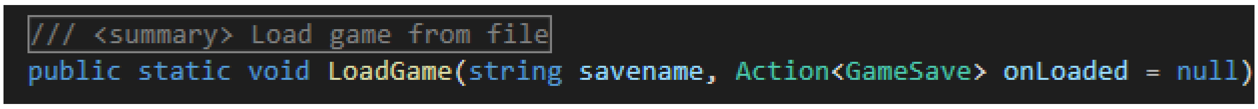
```
/// <summary> Save game into a file
public static void SaveGame(GameEntity game, string savename, Action onSaved = null)
```

Also, you could specify an *Action* which will be played after save process will be finished (for example disable UI save icon)

Inside SaveGame method, *GameEntity* will be serialized to an array of bytes, at first. Then received array will be written to a file inside **<Application.persistentDataPath>/Saves** folder.

Load

To load the game, you should pass a name of a savefile.



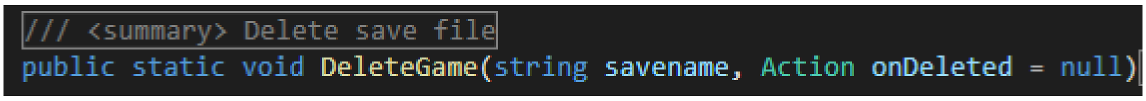
```
/// <summary> Load game from file
public static void LoadGame(string savename, Action<GameSave> onLoaded = null)
```

Also, you could specify an *Action* which will be played after load process will be finished (for example disable UI load icon)

LoadGame operates in reverse order in comparison to SaveGame method. At first, it reads bytes from the file. Then it deserializes bytes to an instance of the *GameSave* class.

Delete

DeleteGame method is the simplest one. It just deletes the file with the specified name.



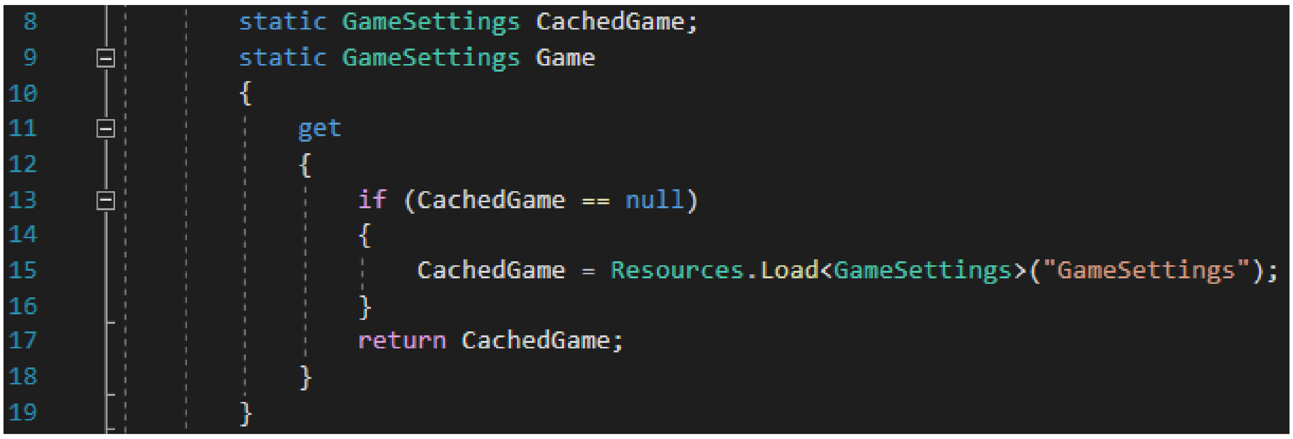
```
/// <summary> Delete save file
public static void DeleteGame(string savename, Action onDeleted = null)
```

Join our Discord

Game Settings

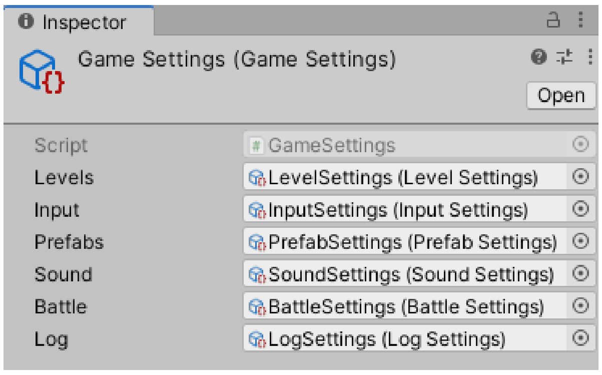
As mentioned above, there is a root data object named *GameSettings*– an instance of *GameSettings* class.

It specifies how all settings will be loaded. From the screenshot below it can be seen that *GameSettings* asset should be inside Resources folder and is loaded when it is called for the first time.



```
        static GameSettings CachedGame;
        static GameSettings Game
        {
            get
            {
                if (CachedGame == null)
                {
                    CachedGame = Resources.Load<GameSettings>("GameSettings");
                }
                return CachedGame;
            }
        }
```

This class is frequently used and to simplify reference calls (something like GameSettings.Instance.Levels) *StaticSettings* class was created.

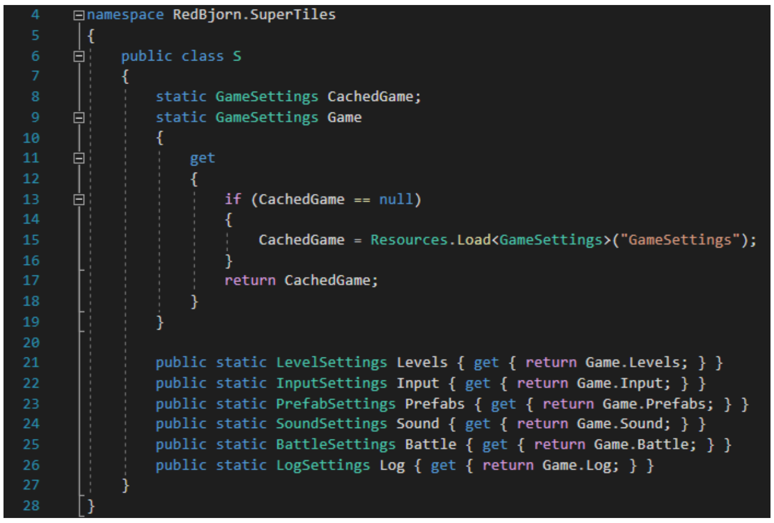


Inspector

Game Settings (Game Settings)

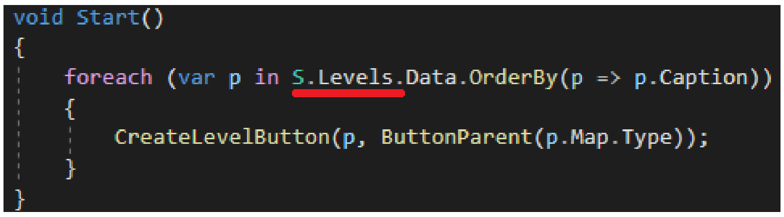
Open

Script	GameSettings
Levels	LevelSettings (Level Settings)
Input	InputSettings (Input Settings)
Prefabs	PrefabSettings (Prefab Settings)
Sound	SoundSettings (Sound Settings)
Battle	BattleSettings (Battle Settings)
Log	LogSettings (Log Settings)



```
namespace RedBjorn.SuperTiles
{
    public class S
    {
        static GameSettings CachedGame;
        static GameSettings Game
        {
            get
            {
                if (CachedGame == null)
                {
                    CachedGame = Resources.Load<GameSettings>("GameSettings");
                }
                return CachedGame;
            }
        }

        public static LevelSettings Levels { get { return Game.Levels; } }
        public static InputSettings Input { get { return Game.Input; } }
        public static PrefabSettings Prefabs { get { return Game.Prefabs; } }
        public static SoundSettings Sound { get { return Game.Sound; } }
        public static BattleSettings Battle { get { return Game.Battle; } }
        public static LogSettings Log { get { return Game.Log; } }
    }
}
```

For example, if you want to get LevelSettings somewhere in the code, you should call S.Levels. S for the



```
void Start()
{
    foreach (var p in S.Levels.Data.OrderBy(p => p.Caption))
    {
        CreateLevelButton(p, ButtonParent(p.Map.Type));
    }
}
```

first character of *StaticSettings*.

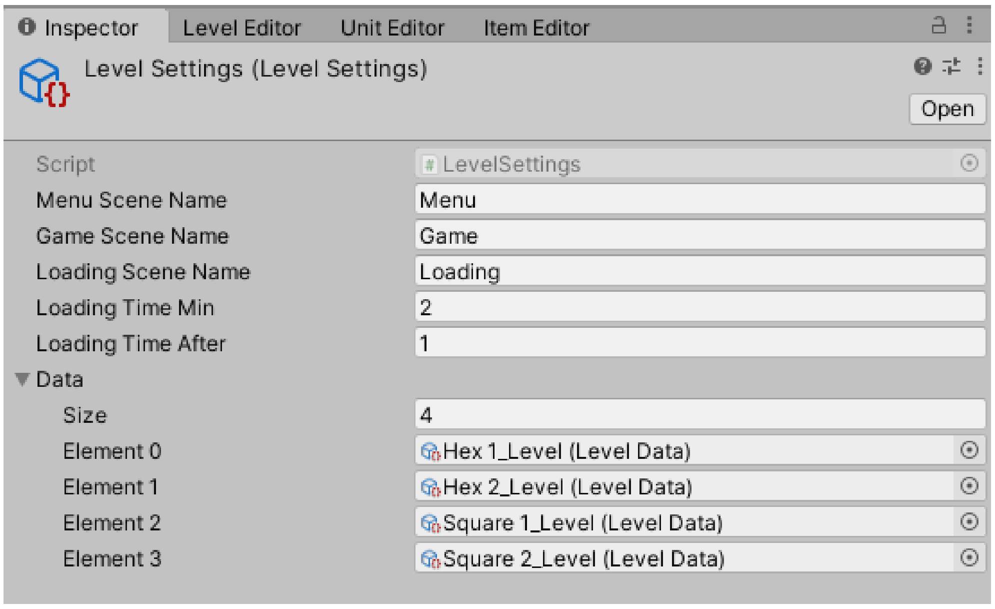
Join our Discord

LevelSettings

Contains data about helper scene names and list of gameplay LevelData

Loading Time Min is a minimal load level duration to avoid fast frame switching when lightweight scene is loading

Loading Time After is a duration between disabling loading screen and enabling game controls.

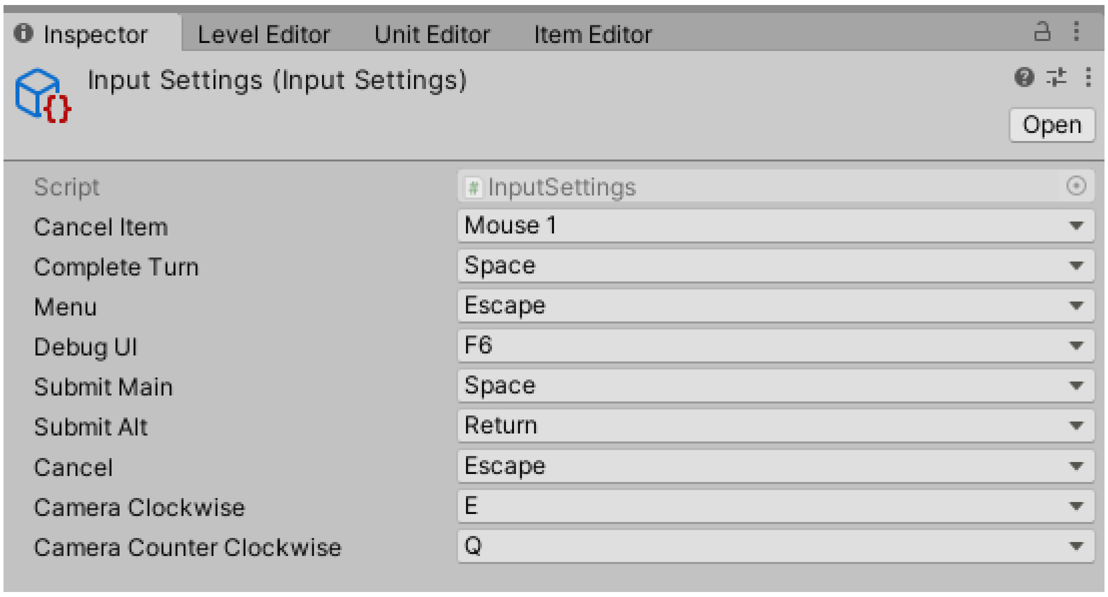


Script	LevelSettings
Menu Scene Name	Menu
Game Scene Name	Game
Loading Scene Name	Loading
Loading Time Min	2
Loading Time After	1
Data	
Size	4
Element 0	Hex 1_Level (Level Data)
Element 1	Hex 2_Level (Level Data)
Element 2	Square 1_Level (Level Data)
Element 3	Square 2_Level (Level Data)

Join our Discord

InputSettings

Contains data about keycodes which is paired to corresponding input actions



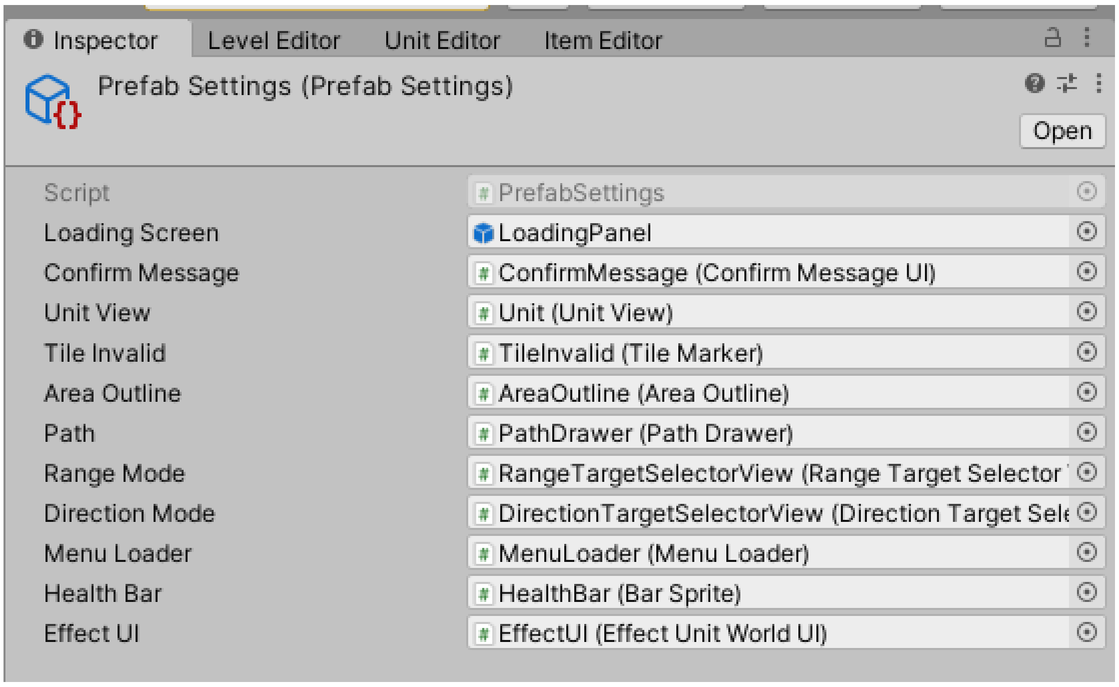
Input Settings (Input Settings)

Script	InputSettings
Cancel Item	Mouse 1
Complete Turn	Space
Menu	Escape
Debug UI	F6
Submit Main	Space
Submit Alt	Return
Cancel	Escape
Camera Clockwise	E
Camera Counter Clockwise	Q

Join our Discord

PrefabSettings

Contains reference to prefabs which do not have own reference holders.

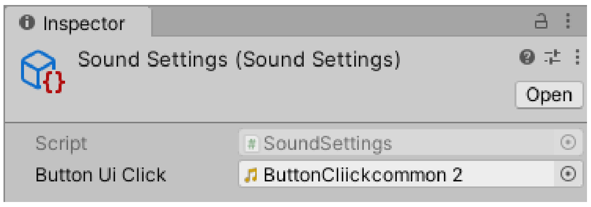


Script	PrefabSettings
Loading Screen	LoadingPanel
Confirm Message	ConfirmMessage (Confirm Message UI)
Unit View	Unit (Unit View)
Tile Invalid	TileInvalid (Tile Marker)
Area Outline	AreaOutline (Area Outline)
Path	PathDrawer (Path Drawer)
Range Mode	RangeTargetSelectorView (Range Target Selector
Direction Mode	DirectionTargetSelectorView (Direction Target Sel
Menu Loader	MenuLoader (Menu Loader)
Health Bar	HealthBar (Bar Sprite)
Effect UI	EffectUI (Effect Unit World UI)

Join our Discord

SoundSettings

Contains reference to used audio clips



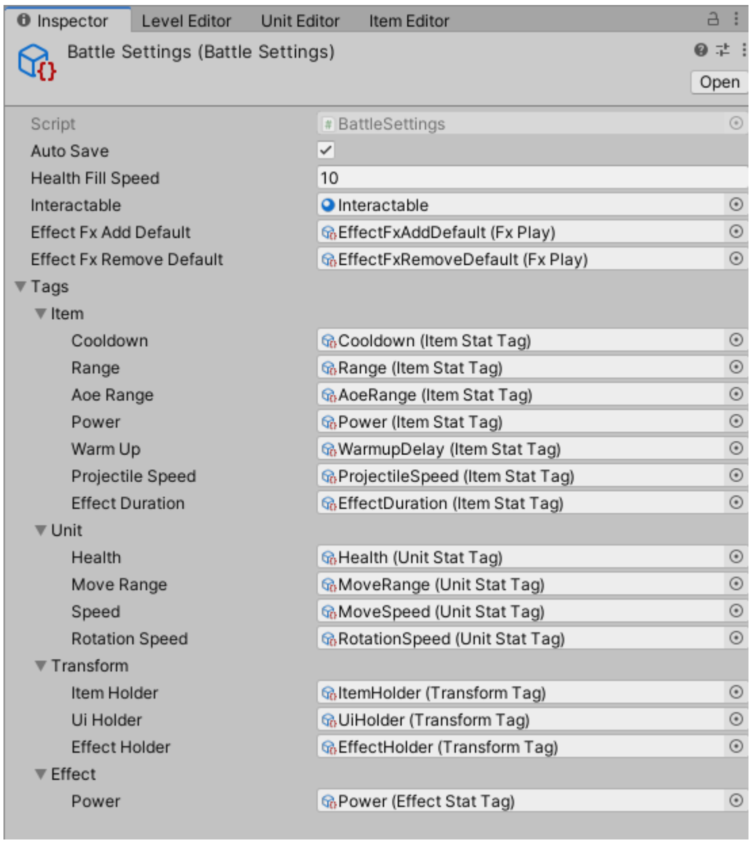
Script: SoundSettings
Button Ui Click: ButtonCliickcommon 2

[Join our Discord]

BattleSettings

Contains information that helps during Battle:

- Could be game autosaved at the end of each turn
- Unit's default interactable material
- Fill Speed of health bar
- Default EffectHandlers which are played when effect will be added (removed)
- Tags mapping

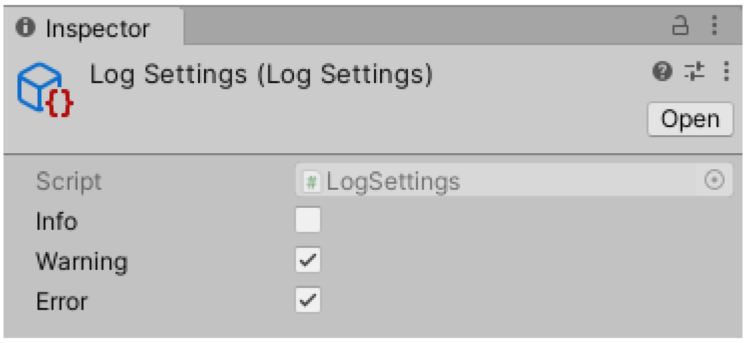


Script	BattleSettings
Auto Save	✓
Health Fill Speed	10
Interactable	Interactable
Effect Fx Add Default	EffectFxAddDefault (Fx Play)
Effect Fx Remove Default	EffectFxRemoveDefault (Fx Play)
Tags	
Item	
Cooldown	Cooldown (Item Stat Tag)
Range	Range (Item Stat Tag)
Aoe Range	AoeRange (Item Stat Tag)
Power	Power (Item Stat Tag)
Warm Up	WarmupDelay (Item Stat Tag)
Projectile Speed	ProjectileSpeed (Item Stat Tag)
Effect Duration	EffectDuration (Item Stat Tag)
Unit	
Health	Health (Unit Stat Tag)
Move Range	MoveRange (Unit Stat Tag)
Speed	MoveSpeed (Unit Stat Tag)
Rotation Speed	RotationSpeed (Unit Stat Tag)
Transform	
Item Holder	ItemHolder (Transform Tag)
Ui Holder	UiHolder (Transform Tag)
Effect Holder	EffectHolder (Transform Tag)
Effect	
Power	Power (Effect Stat Tag)

Join our Discord

LogSettings

Contains Log level toggles



Script	LogSettings
Info	☐
Warning	✓
Error	✓

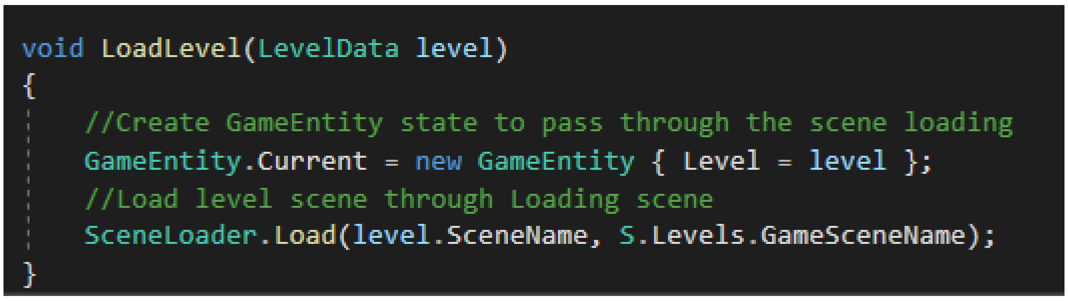
Join our Discord

Scenes

Menu

It is a scene which was created only for one purpose: to select *LevelData* which will be loaded.

The core part is located inside `LoadLevel` method



```
void LoadLevel(LevelData level)
{
    //Create GameEntity state to pass through the scene loading
    GameEntity.Current = new GameEntity { Level = level };
    //Load level scene through Loading scene
    SceneLoader.Load(level.SceneName, S.Levels.GameSceneName);
}
```

First, we create a new *GameEntity* instance with specified *LevelData*. Then the instance is assigned to a static variable to "survive" scene loading process.

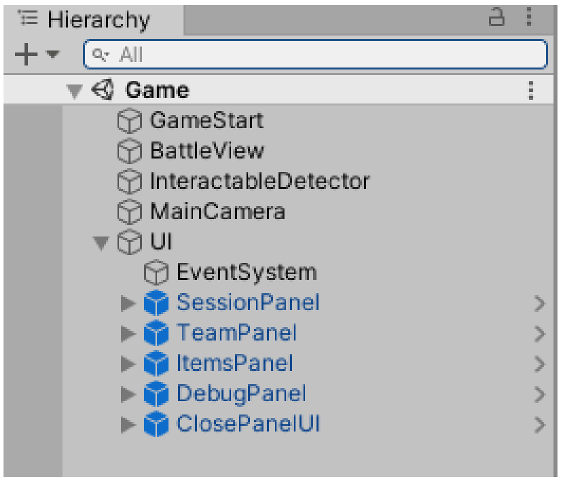
Second, we load selected level with additional Game scene through the Loading scene.

Summarizing the above, Menu scene could be customized in any way. The only thing that needs to be left is only these two lines.

Join our Discord

Game

Game scene includes a bunch of gameObjects which is needed for correct Battle playing.

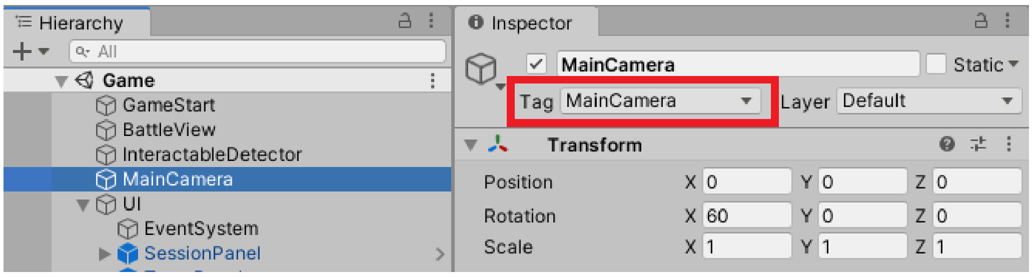


GameStart triggers scene loading and start essential class creating (GameEntity, BattleEntity, UnitEntity, UnitAiEntity and others)

BattleView handles **Player** or **Ai** input and change Battle state respectively

InteractableDetector check If there is any interactable gameObject under the mouse

MainCamera is nothing than camera which renders the game. However, it has Unity standard tag Main Camera

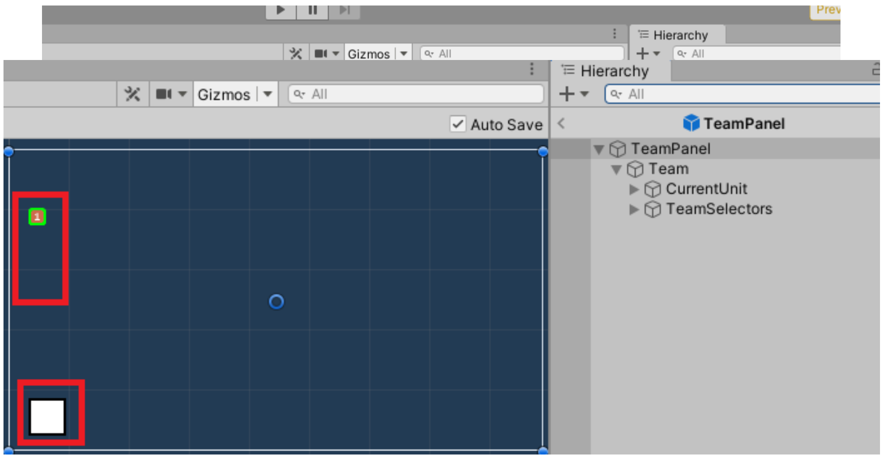


Ui is empty gameObject which is a parent for Ui related gameObjects

EventSystem – gameObject which is needed for correct input handling in Unity input default pipeline

Session panel represents complete turn button and start Battle button (when Auto Start logic is disabled) and two information texts:

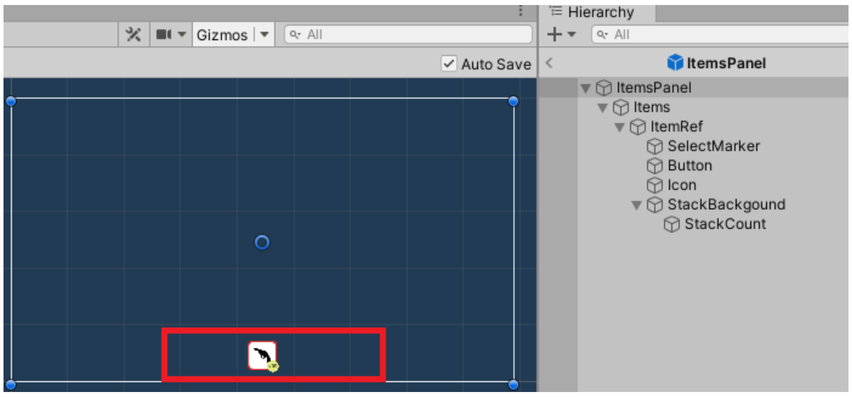
1. Turn Panel shows current turn and Battle state
2. Tooltip shows info about unit and item when mouse is pointing such objects



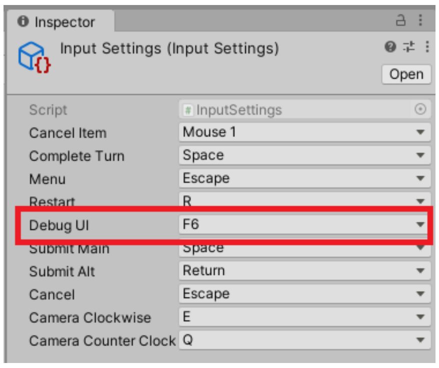
Team Panel shows unit button numbers and selected unit avatar

Join our Discord

Item panel shows all items for selected unit.

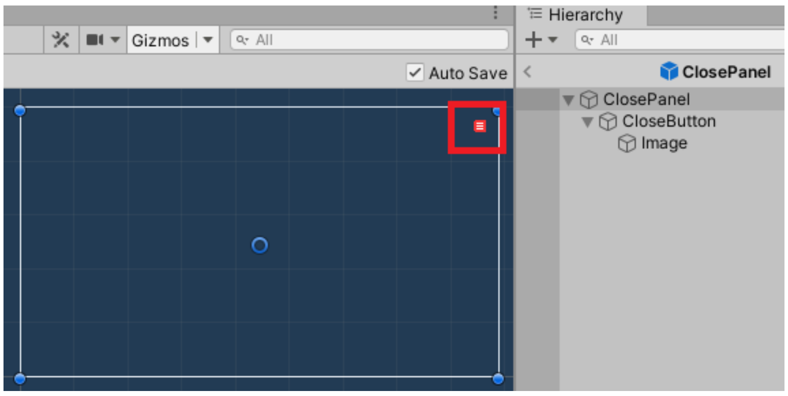


DebugPanel shows debug information about current Battle. It is closed by default and can be opened by clicking Debug UI keycode



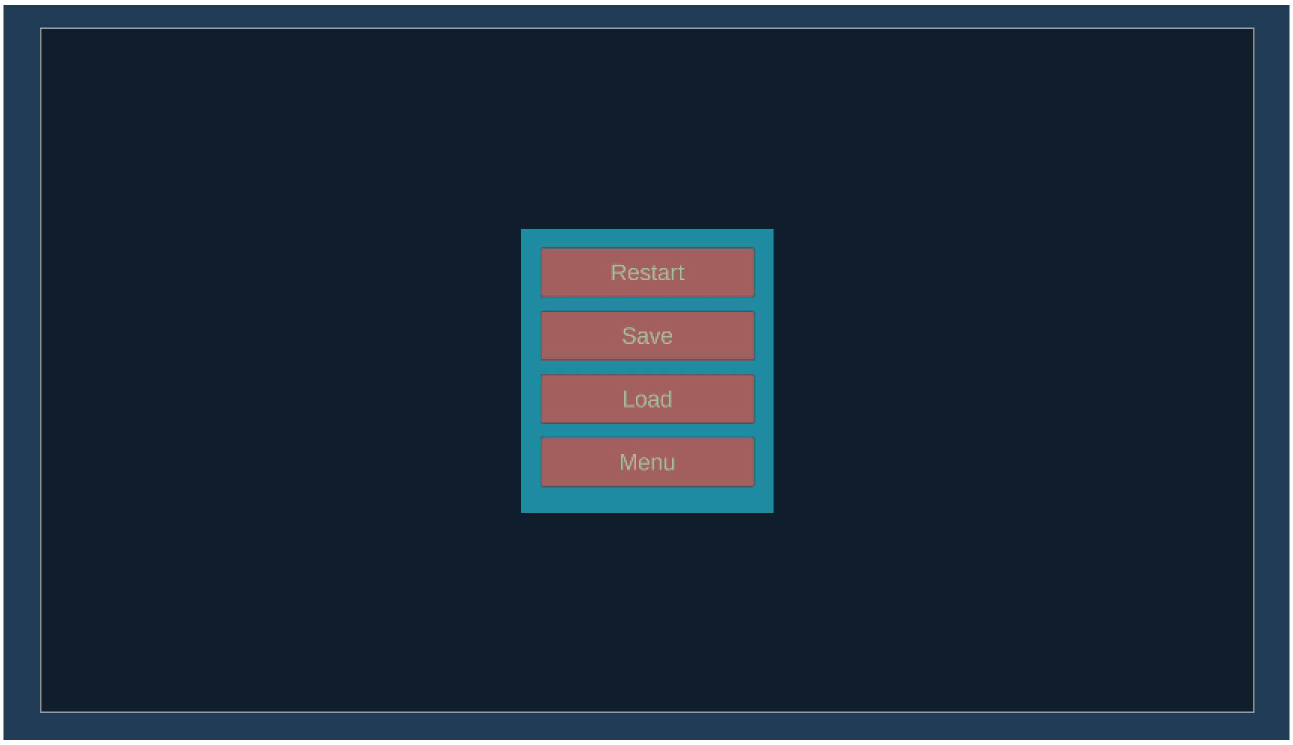
Script	InputSettings
Cancel Item	Mouse 1
Complete Turn	Space
Menu	Escape
Restart	R
Debug UI	F6
Submit Main	Space
Submit Alt	Return
Cancel	Escape
Camera Clockwise	E
Camera Counter Clock	Q

ClosePanel represents button which open MenuPanel



Join our Discord

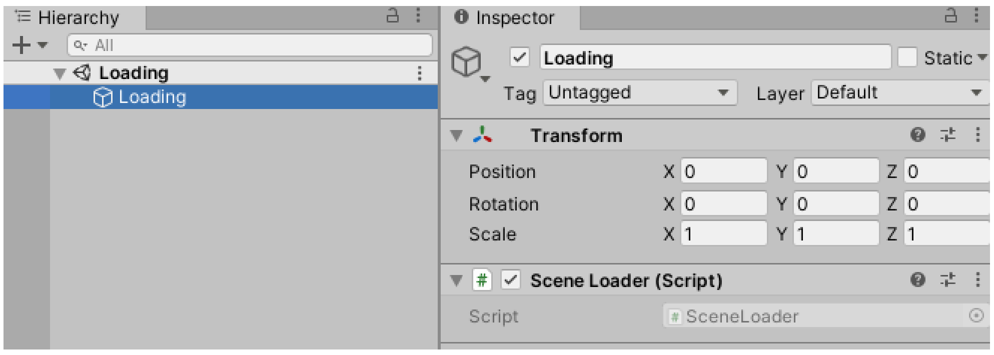
MenuPanel represents in-game menu. Black semitransparent background is a button. Click on this button closes MenuPanel.



- Restart button restarts current level from the beginning
- Save button saves current session to a file
- Load button loads game state from a file
- Menu button moves the game to Menu scene

Loading

Loading scene is a transitional scene between menu and game scenes. At runtime it instantiates LoadingPanel with black background and white rotating circle


Hierarchy
- Loading
  - Loading

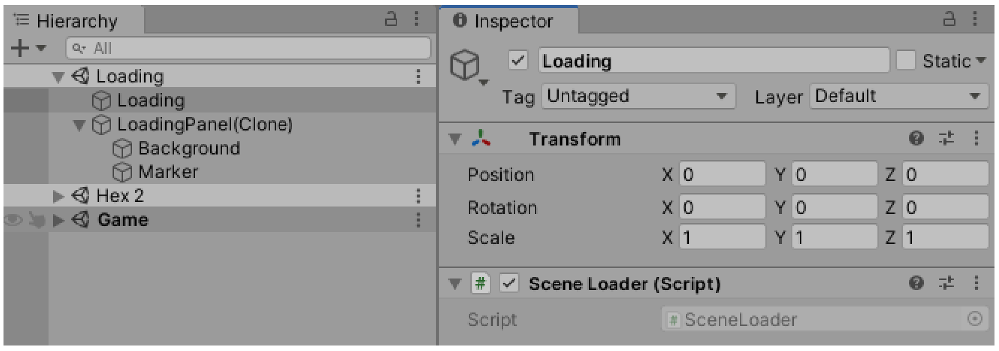
Inspector: Loading — Static — Tag: Untagged — Layer: Default

Transform

	X	Y	Z
Position	0	0	0
Rotation	0	0	0
Scale	1	1	1

Scene Loader (Script) — Script: SceneLoader

Editor


Hierarchy
- Loading
  - Loading
  - LoadingPanel(Clone)
    - Background
    - Marker
- Hex 2
- Game

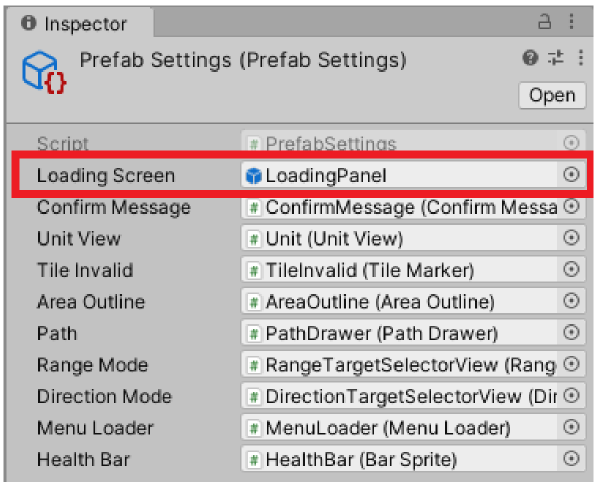
Inspector: Loading — Static — Tag: Untagged — Layer: Default

Transform

	X	Y	Z
Position	0	0	0
Rotation	0	0	0
Scale	1	1	1

Scene Loader (Script) — Script: SceneLoader

Playmode


Inspector — Prefab Settings (Prefab Settings) — Open

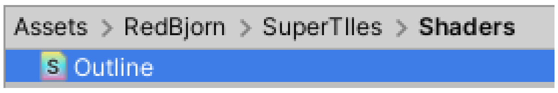
Script	PrefabSettings
Loading Screen	LoadingPanel
Confirm Message	ConfirmMessage (Confirm Messa
Unit View	Unit (Unit View)
Tile Invalid	TileInvalid (Tile Marker)
Area Outline	AreaOutline (Area Outline)
Path	PathDrawer (Path Drawer)
Range Mode	RangeTargetSelectorView (Rang
Direction Mode	DirectionTargetSelectorView (Dir
Menu Loader	MenuLoader (Menu Loader)
Health Bar	HealthBar (Bar Sprite)

The main logic is incapsulated inside SceneLoader class and handles the order of scene loading.

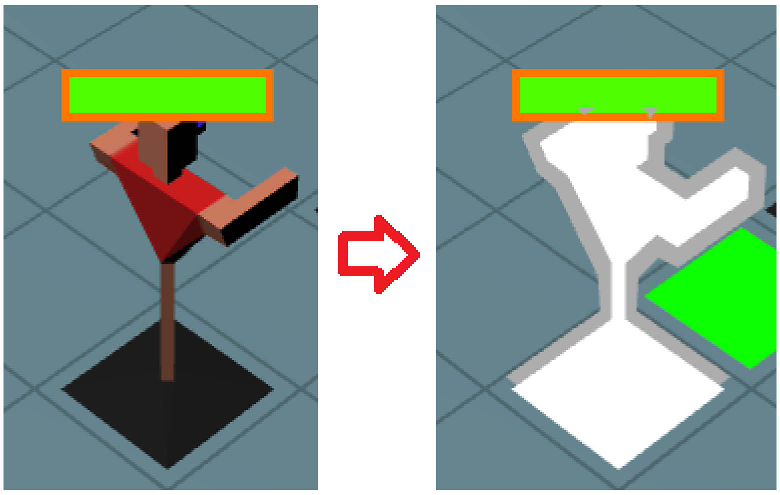
Join our Discord

Shaders

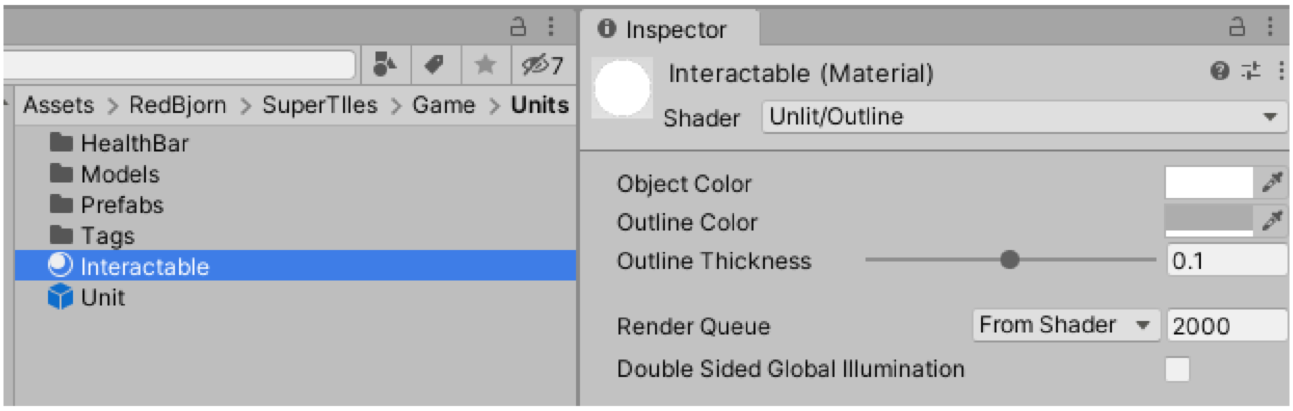
There is only one shader: Outline. It represents straightforward implementation of outline logic; however, it does its work as expected.


Assets > RedBjorn > SuperTIles > **Shaders**

Outline



Interactable material is a sole use of current shader.


Assets > RedBjorn > SuperTIles > Game > **Units**

- HealthBar
- Models
- Prefabs
- Tags
- Interactable
- Unit

Inspector

Interactable (Material)

Shader: Unlit/Outline

Object Color	
Outline Color	
Outline Thickness	0.1
Render Queue	From Shader 2000
Double Sided Global Illumination	☐

Join our Discord