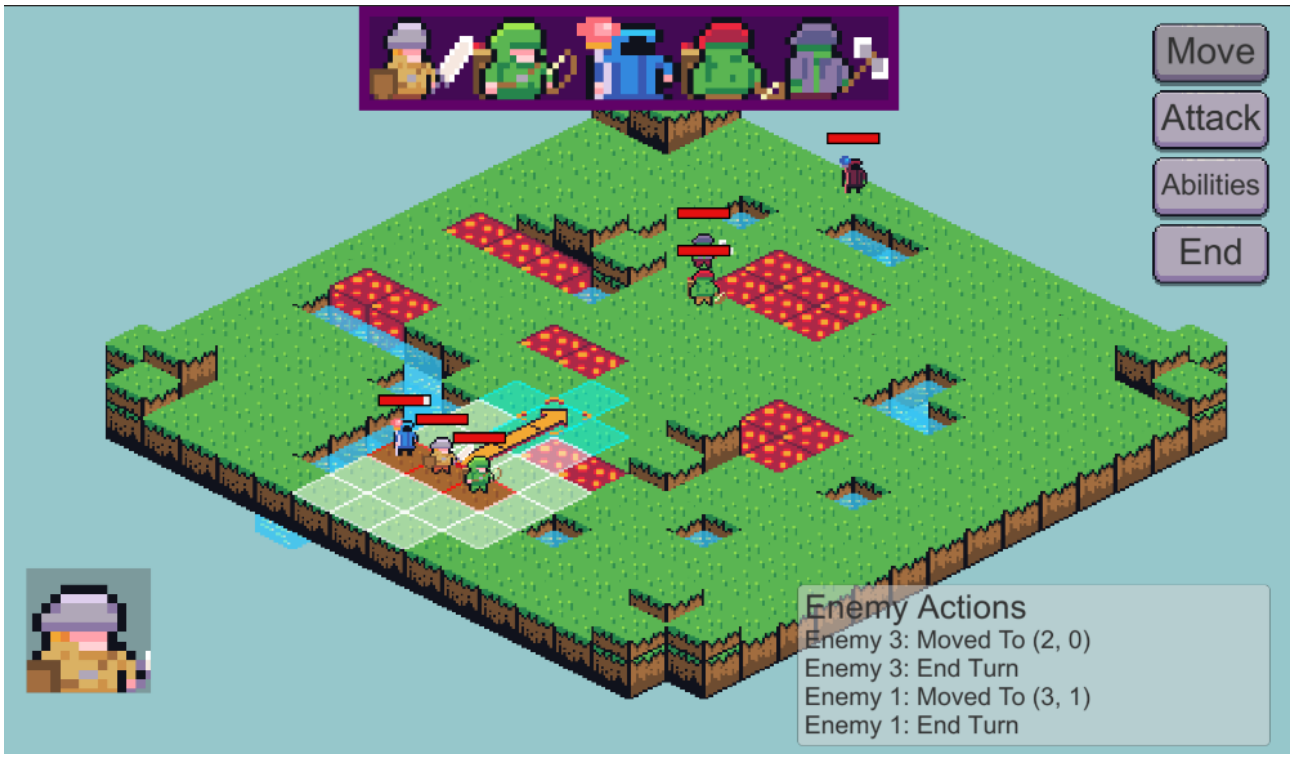
Introduction

Tactics Toolkit is an all-in-one template for developing SRPG games. This template is the result of the last year I spent creating my own SRPG game. Taking all the knowledge I've gained and distilling it into seven different scenes, featuring A* Pathfinding, RPG Character Systems, Effect Tiles, Turn-Based Mechanics, Attack Systems, Ability Systems, and Enemy AI. This template is also in constant development as I develop my own game and systems, and I will add improvements I made to this project for your use too.



This project was designed with fast prototyping in mind. Within it you'll find that no system has a dependency on another aswell as dynamic and customisable scriptable objects. This makes for a level of flexibility that is essential for generating fast prototypes.

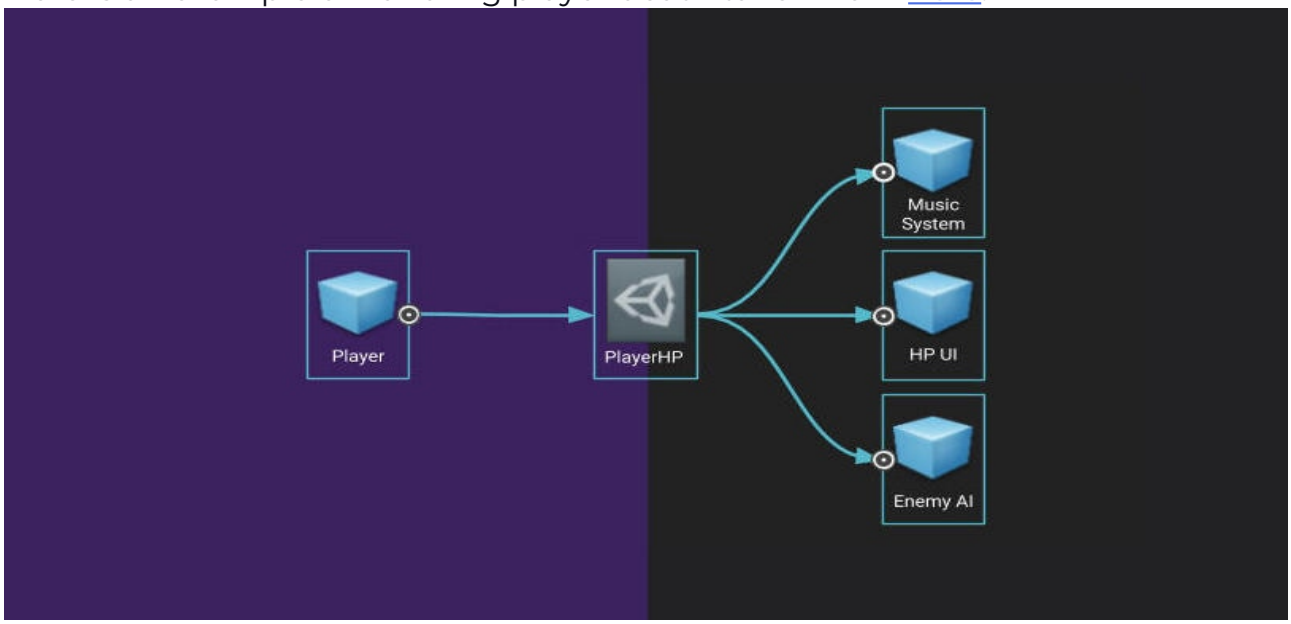
How to go forward

Once you've familiarised yourself with the project. Begin stripping it down and changing values to fit the design of your game. There is no one meets all when it comes to strategic rpgs. Every game must have it's own feel and flow. So I would encourage you to take these examples and expand apon them to fit the feel of the game you want to make. A good place to start and get your toes wet would be the Character class systems. It's quite flexible and can drastically change the feel of the game depending on how you change them.

Event System

The project's core is all developed using a Scriptable Object Event System. Inspired by a 2017 GDC talk, this event system allows the decoupling and the removal dependencies.

Here is an example of handling player death taken from here.

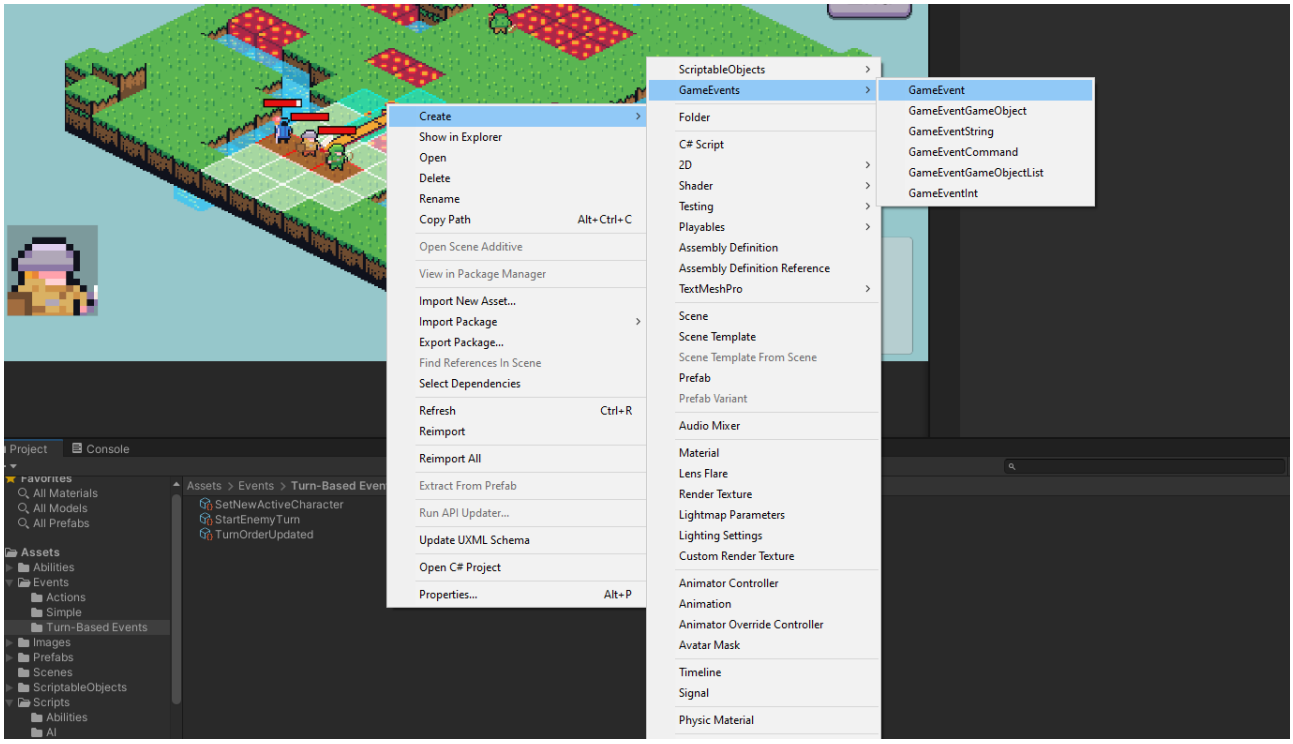


Without the event system, the Music System, Game Over UI, and Enemy would depend on the Player. If you wanted to make a test scene for any of these, you would also need to add a Player to that scene and include whatever dependencies the Player has, i.e., a MovementController—making it more and more challenging to create these simple tests. With the event system, when the Player dies, the OnPlayerDied event is raised, and the Music System, Game Over UI, and Enemy listen for it and execute. The Music System doesn't care if the Player, Game Over UI, or Enemy exists within the scenes hierarchy. It also doesn't need to constantly check the Player's status within the Update loop, making it much more straightforward.

How to use the Event System

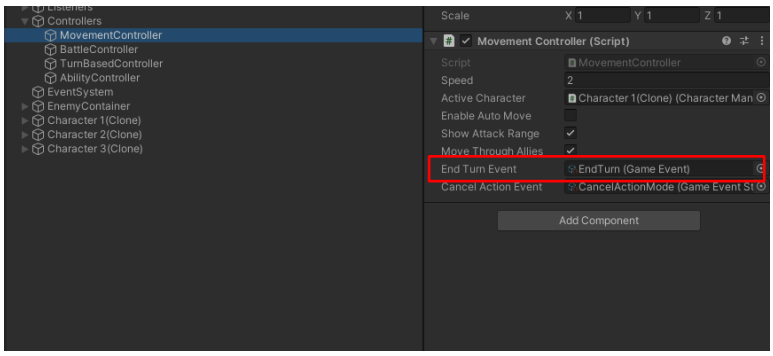
The event types I use are in Scripts->Scriptable Objects->Events.

All the used events are in the Events folder. To create a new one right, click within the folder, go to create-> Game Events, and make the event based on the type you want.



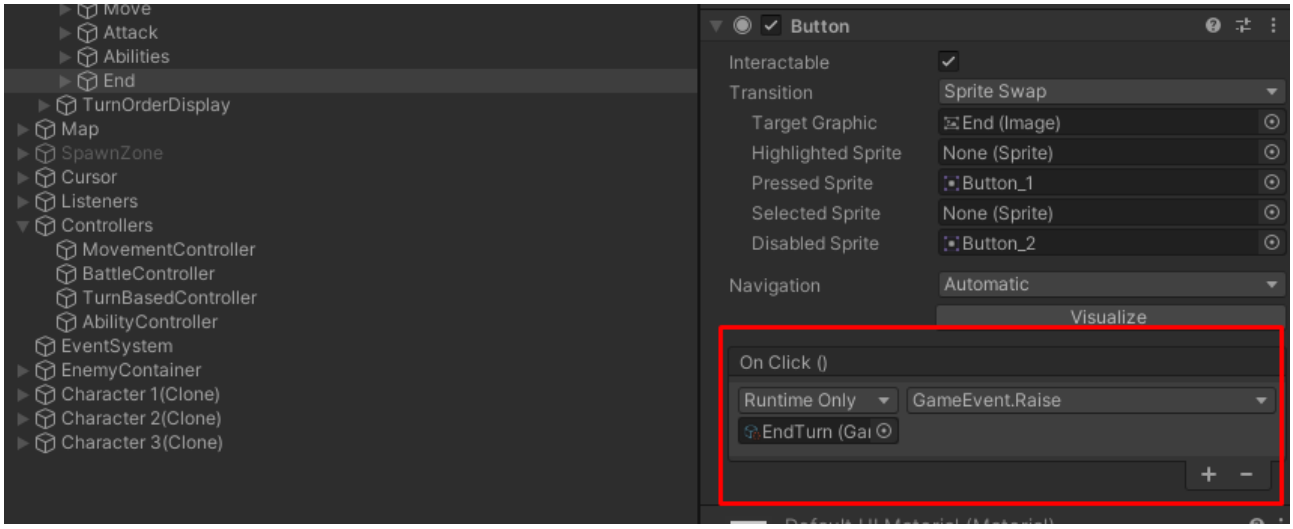
Then add that event to where ever it should be raised, be it a script or button.

Script Example

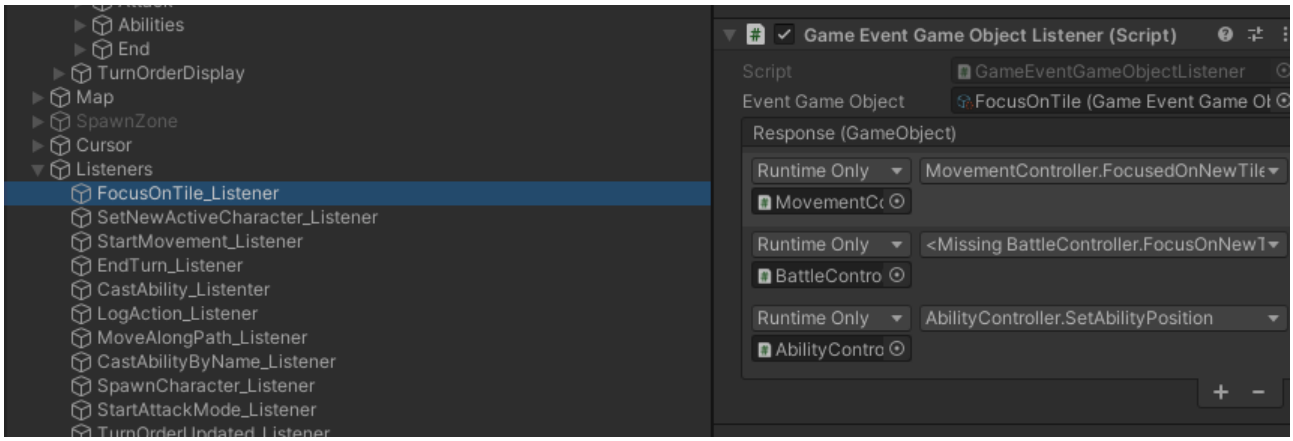



```
if (enableAutoMove)
{
    if(endTurnEvent)
        endTurnEvent.Raise();
    else
        SetActiveCharacter(activeCharacter.gameObject);
}
```

Button Example

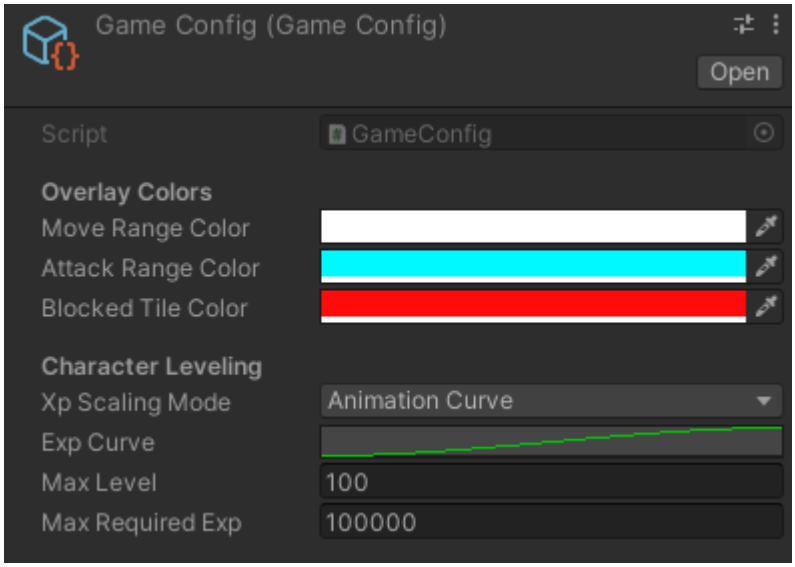


Next, you must create a listener to receive the event and execute its corresponding function. Make sure the listener type matches the game event type. i.e., A GameEventInt needs a GameEventIntListener.



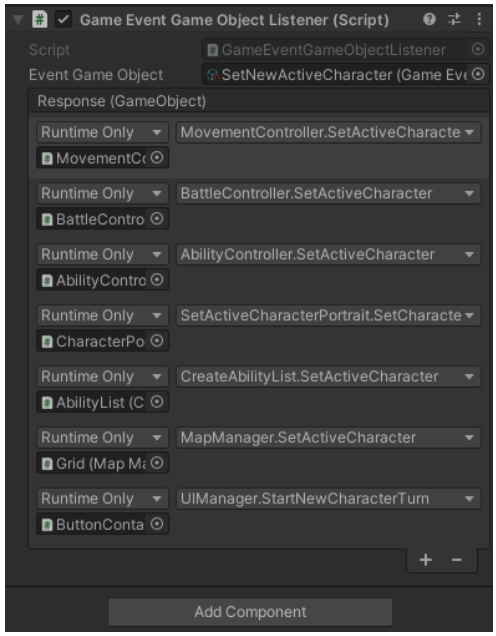
Game Config

The GameConfig is a Scriptable object that contains some constants that the game uses. Technically this doesn't need to be a Scriptable object, but I thought it was fitting. A few scripts require this, including the OverlayController and all Entities.



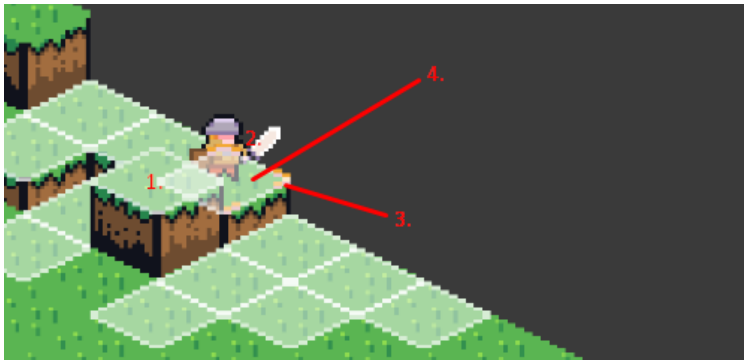
Active Character

The ActiveCharacter is a variable used throughout the project. When a new character's turn begins, an event is triggered that sets the ActiveCharacter in every Controller.

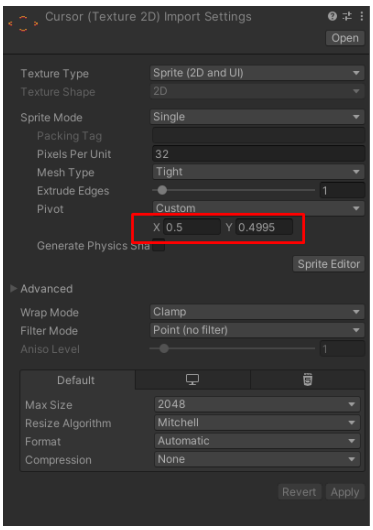


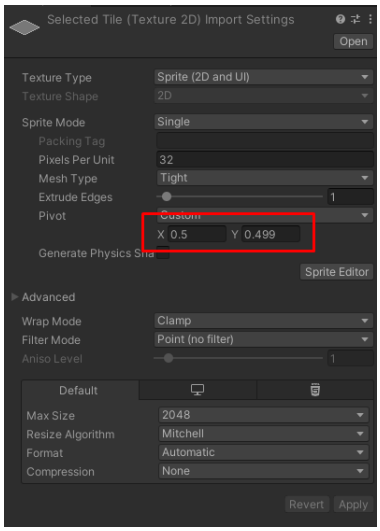
Sprite Layering

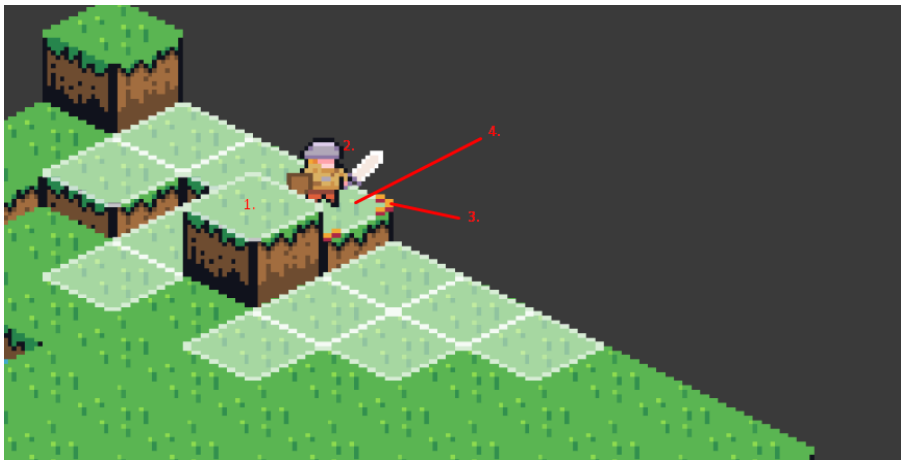
By default Unity will layer images based on the z axis. However is certain instances that there can be multiple image on the same z axis and the rendering order can be incorrect.



This can be easily fixed by applying small offsets using custom pivot points. Another option is applying the offsets progmatically.

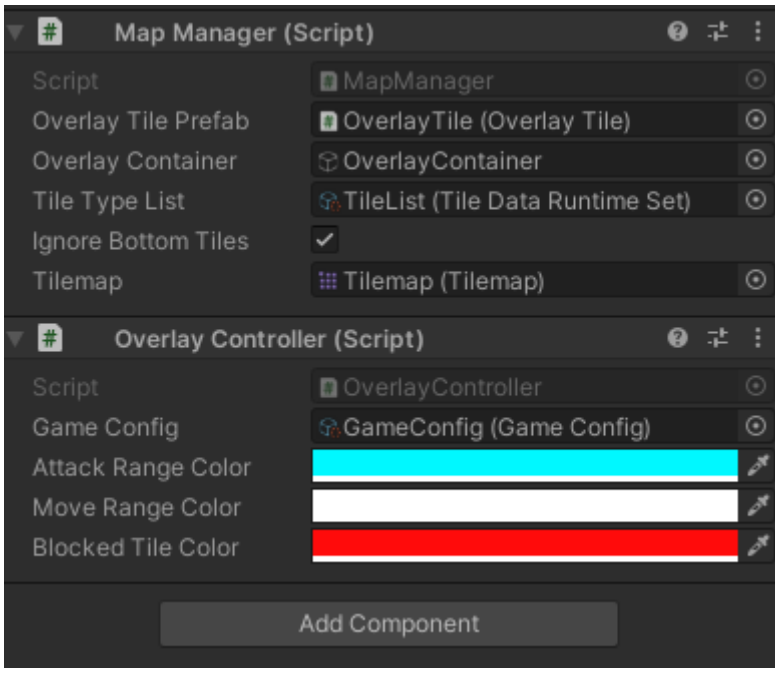




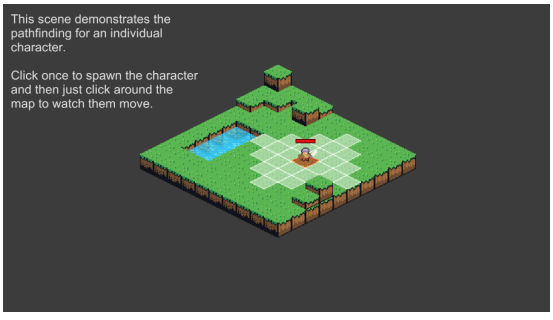


Map And OverlayTiles

This project uses an isometric tilemap, but it should theoretically work with a top-down tilemap too. The way it works is that when the scene starts, the MapManager will loop through the tilemap and create an overlay tile prefab over every tile. In addition, there is an option for ignoreBottomTiles. This option will automatically set tiles at z position 0 to be blocked and non-traversable. Is this project, for example, that is the water tile. The map manager is also a singleton that holds a dictionary for every tile within the level and a few methods for interacting with those tiles—most notably, finding the neighboring tiles of a specific location.

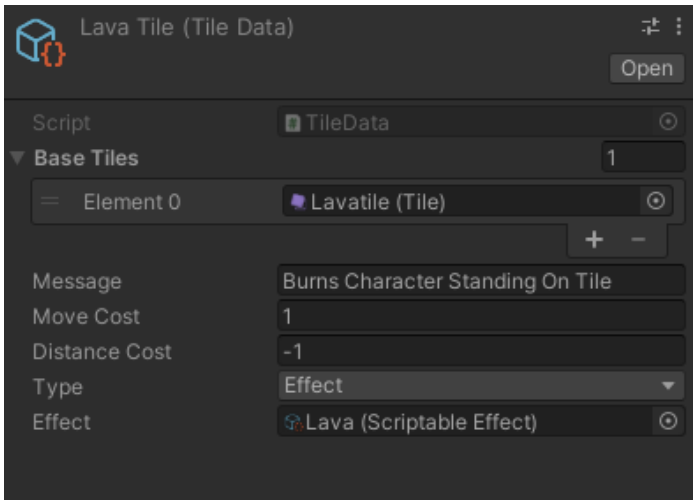


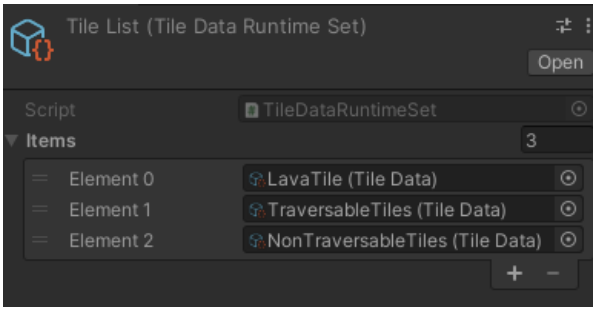
By default, the image of an overlay tile is invisible. But by using the preset colors on the OverlayController, you can color them as needed. For example, in the MovementController, when the movement mode is enabled, it gets all of the tiles within the characters movement range and colors them with the MoveRangeColor(white)



Tile Data

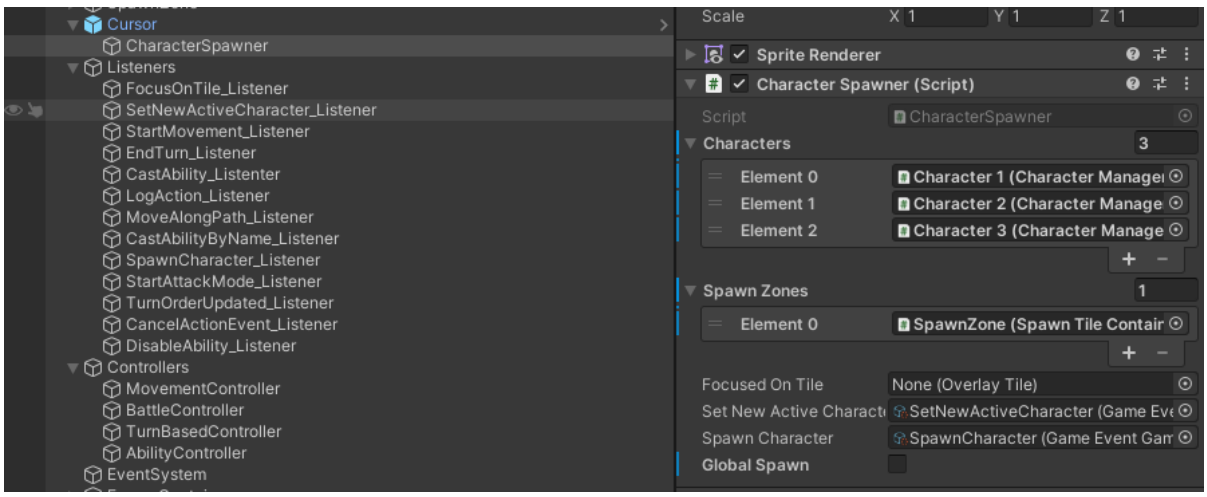
At the moment, TileData is a Scriptable Object that is a relatively underused resource within the project but has some powerful potential. TileData is used for attaching Effects (see more on this below) to the OverlayTile when applicable, but this will be expanded in the future. TileData contains a list of BaseTiles used within the Tilemap Pallete. While the MapManager generates the overlay tiles, it checks which tile is used and attaches the correct TileData to that OverlayTile.



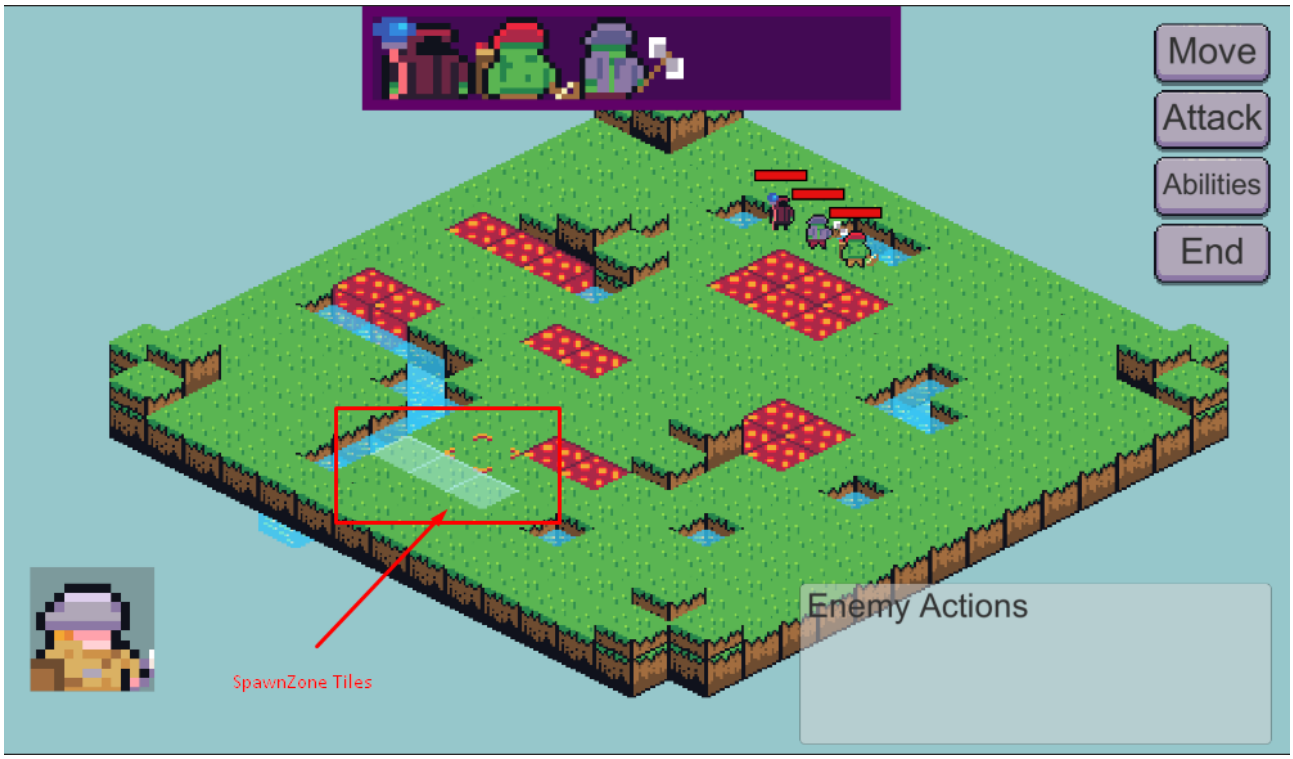


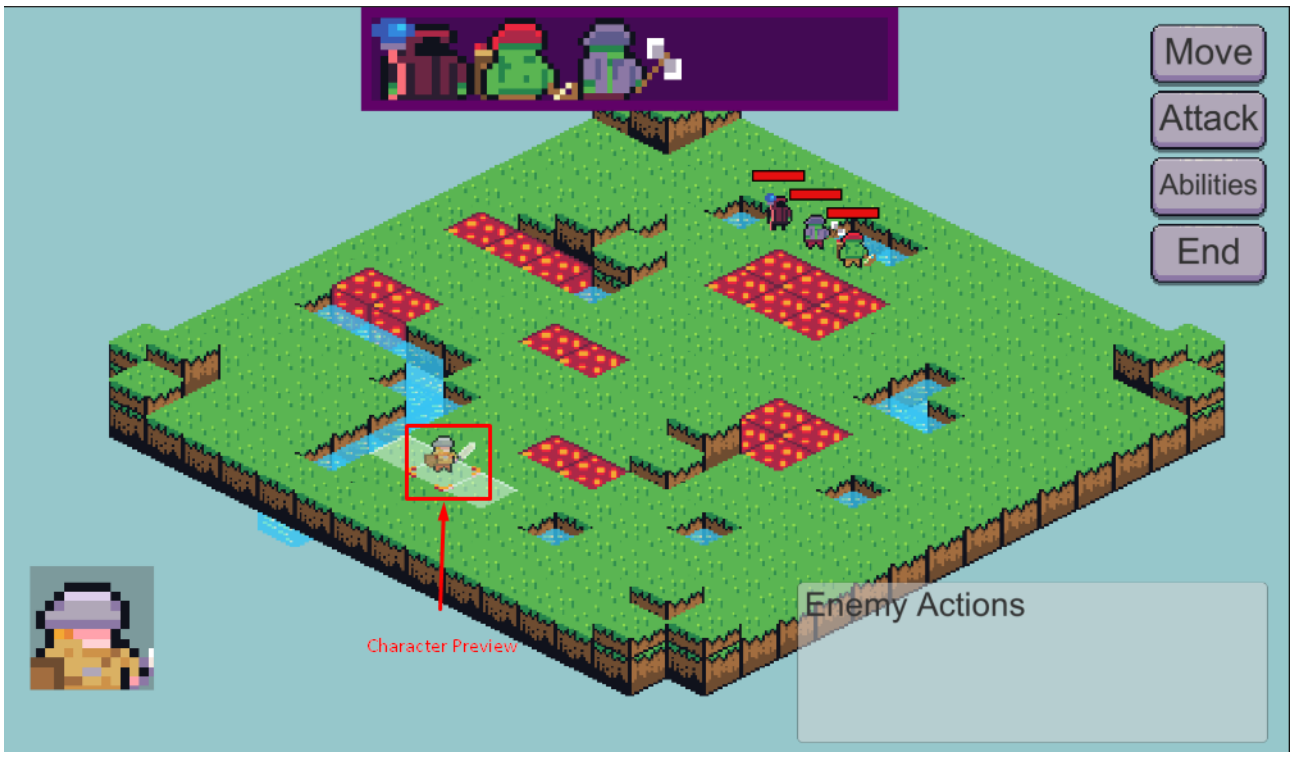
Spawning a Character

Within the Cursor GameObject, there is a CharacterSpawner. This controls spawning in characters assuming they're not already on the screen. The script contains two lists, a Characters List and a SpawnZones list. There is also an option for GlobalSpawn. If GlobalSpawn is enabled, you can spawn a character at any available tile on the map, and it will ignore tiles within the SpawnZone list.



A SpawnZone is a GameObject containing a list of tiles in which a character can spawn—using the character teamID; it checks if a character can spawn here and shows a preview if it can. When you click on the tile, the character will spawn.

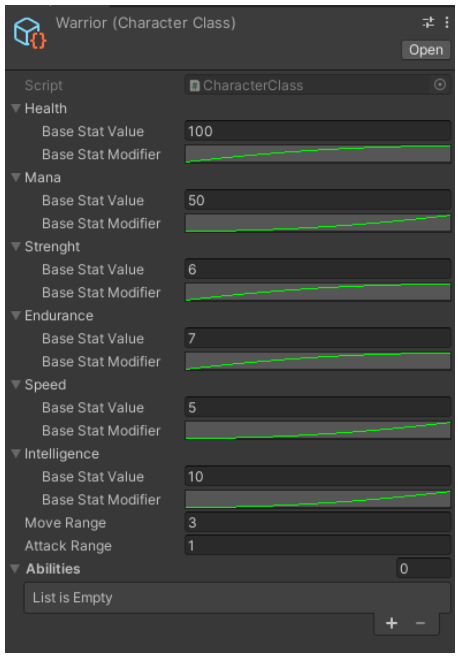




RPG Character System Example

The RPG character systems in this project are an example. They can and should be expanded upon to fit the game you wish to make.
A character/entity starts with a character class. The character class examples can be

A Character Class is a scriptable object containing a list of base attributes/stats and AnimationCurves used to weight each attribute's random scaling.



When a character levels up, a random number between 1 and 10 is added to each stat. This random number is weighted using an animation curve.

For example, the Strength Stat is weighted with the Warrior class, so on average, the stat increase will be closer to 10 than 0 most of the time. In contrast, the Speed stat is weighted in the opposite direction.

The other half of the character system is level exp scaling, how the exp scales is contained within the GameConfig scriptable object.
There are three examples of exp scaling: Constant, Disgaea, and Animation Curve.

1. Constant means the required exp to level up will increase the same amount at every level.

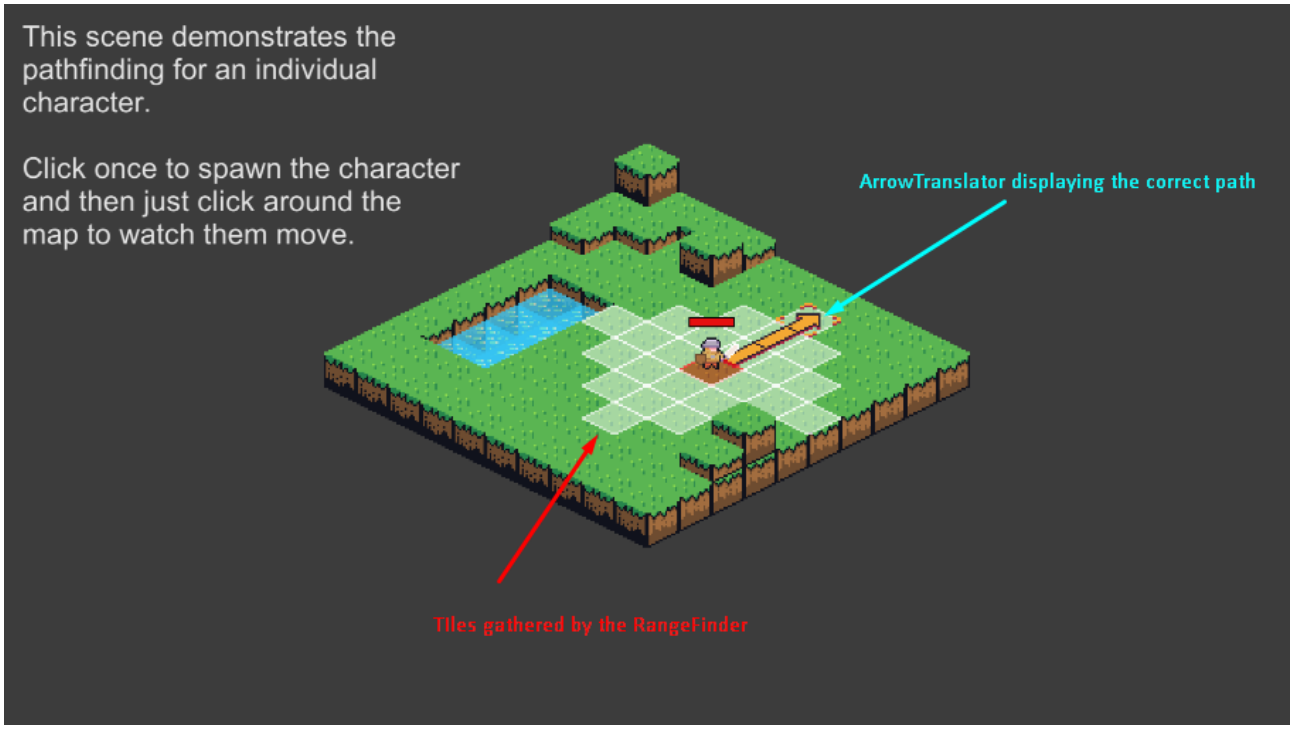
2. Disgaea is the same formula that the Disgaea series uses for its level scaling.

3. Animation Curve uses an animation curve plus two other values to determine how it scales the Max Level and Max Required Exp. (Max Required Exp will be the exp required to reach the final level)

Character Movement

This project uses an A* Pathfinding algorithm to find an optimal path when traveling from A to B. The MovementController executes all movements for both Player Characters and Enemy AI. Although both access the Controller in different ways.

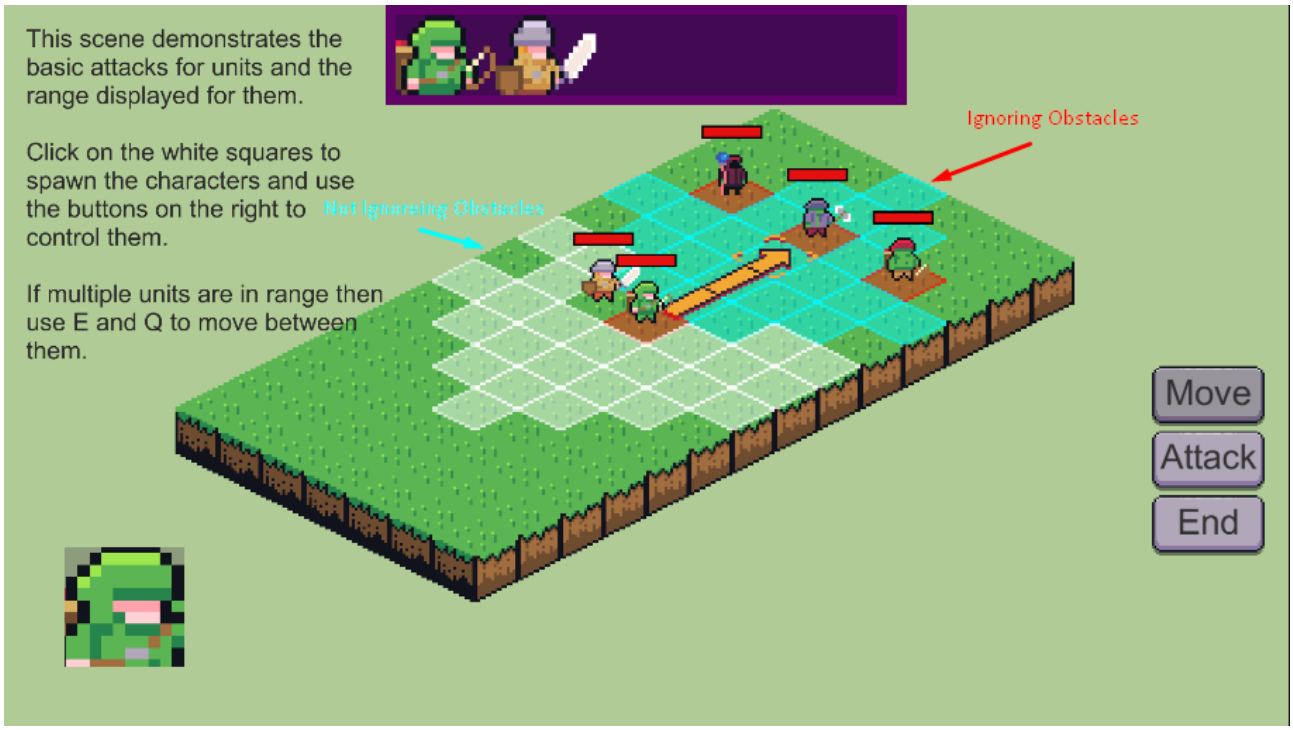
For the Player Characters, the Move button tells the Controller to enter MovementMode. Once MovementMode is enabled, the RangeFinder class immediately finds all the inMovementRange tiles for the Active Player and colors them in the MoveRangeColor (white). Next, if you mouse over any InRangeTiles, the Pathfinder class creates a new Path and gives it to an ArrowTranslator, which takes a list of positions and calculates which path image to use at every tile. Next, once the Player clicks on an InRangeTile, the MovementController will move the character along that path until it reaches its location. While calculating the path, there is also an option to display the character's attack range (blue). The attack range shows the user who/what they can attack before moving to that location. The Range Finder also gets the tiles for this attack range.



For the Enemy AI, the MovementController is slightly less involved. On an enemy's turn, an event sends a predefined path to the MovementController, and it will automatically begin moving.

Range Finder

The rangefinder is used frequently throughout the project. It takes an origin position on the map and an integer representing the Range in which we want to find the tiles. It also has some booleans that make the function a bit more flexible. Mainly whether or not to ignore obstacles. For example, when getting all the tiles in Range for movement, you wouldn't want to ignore obstacles since a character cannot move where the tile is blocked. But you would like to ignore those obstacles for an attack since a ranged attack might go over those blocked obstacles.



Arrow Translator

The arrow translator is a series of if statements determining which image should be placed on a tile. Then, the image's direction is calculated by checking the past and future directions.

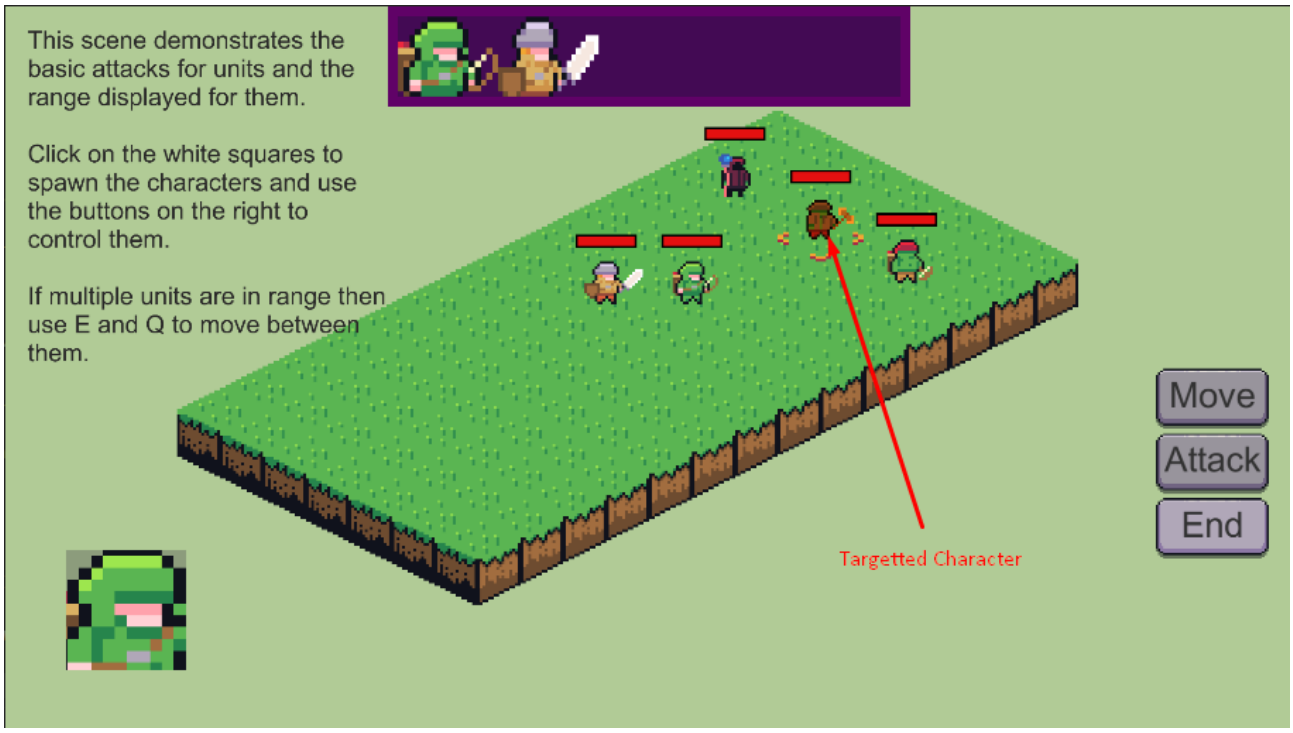
```
bool isFinal = futureTile == null;

Vector2Int pastDirection = previousTile != null ? currentTile.grid2DLocation - previousTile.grid2DLocation : new Vector2Int(0, 0);
Vector2Int futureDirection = futureTile != null ? futureTile.grid2DLocation - currentTile.grid2DLocation : new Vector2Int(0, 0);
Vector2Int direction = pastDirection != futureDirection ? pastDirection + futureDirection : futureDirection;
```

It then returns an enum that is mapped to a list of images held within the OverlayTile Prefab.

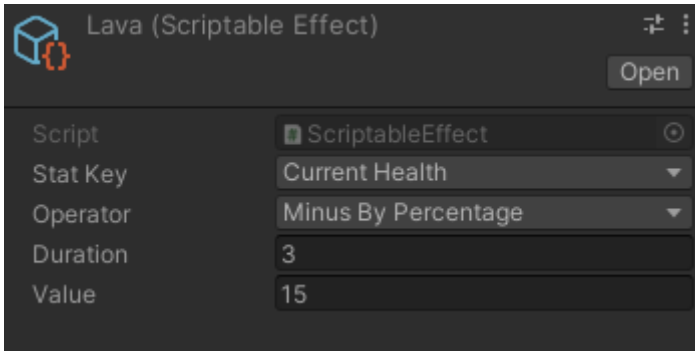
Basic Attacks

The BattleController controls the primary attack command. When the user presses the attack button, Attack Mode is enabled. Then all of the entities within the active character's attack range are gathered, and the first one in the list becomes targeted. From there, the targetted entity will begin changing its color from normal to red. You can then use E and Q to switch between targets and press space to damage the targetted character. Damage is calculated using the active character strength stat and the targeted characters endurance stat from within the entity class. I would encourage you to experiment with this and change it to fit the game you wish to make. After that, there is a check to see if the targeted character should die, and a simple camera shake is executed too.



Effects

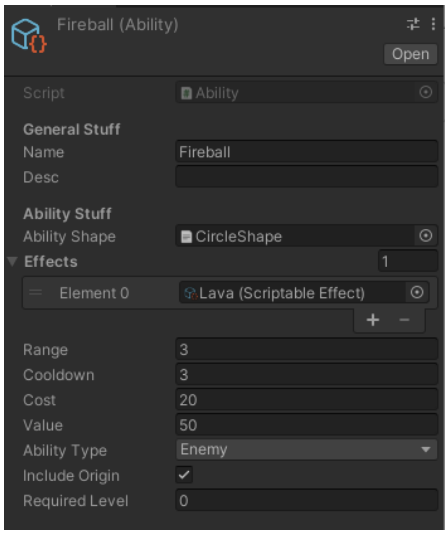
The effects or ScriptableEffects are scriptable objects used in both Tiles and Abilities. With it, you can choose which stat/attribute it will change, how it will change, and for how long that change will take place. So the scriptable effect object can support things like burn effects, debuffs, and buffs.



When a character ends, turns on an effect tile, or is hit by an ability with an effect attached, a stat modifier is attached to that character's specific stat. Then when that character's turn starts again, the stat modifier is allied until the duration is completed. Then that stat is returned to normal except for current health. Also, if the duration of an effect is set to 0, then the effect is applied instantly.

Abilities

Abilities are ScriptableObjects that are attached to character classes. For example, the ability SO contains a list of Effects, a Range, a Cooldown, a Cost, a Value, an Ability Type, and an option to include the origin, a required level value, and an ability shape.



1. I explained the list of effects in the previous heading. When a character is hit with an ability, all the **effects** are applied to the character's stats.

2. The **Range** means how far away a character can cast the ability.

3. **Cooldown** is how many turns the character must wait before they can cast it again.

4. The **Cost** is how much mana is required to cast the ability.

5. The **Value** is the based damage value the ability will deal.

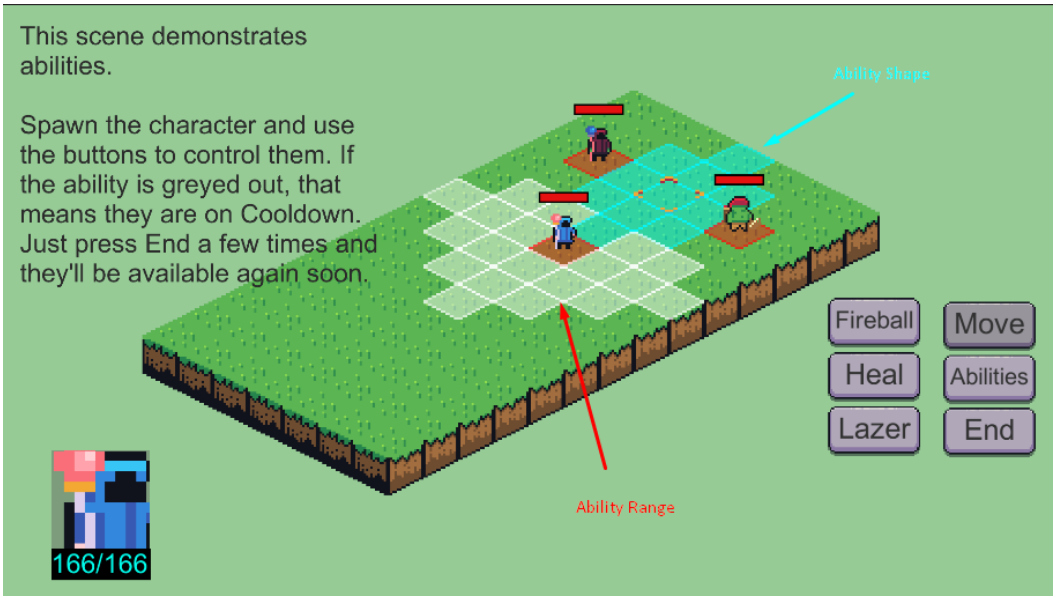
6. The **Ability Type** determines whom the ability can target: Enemies, Allies and All.

7. **Include Origin** determines if the origin of the ability is included in the casting. Sometimes the source may be the caster's location, so it's good to be able to ignore that if needed.

8. The **Required Level** is used to confirm if the character can use the ability based on level.

9. Finally, the **Ability Shape** is a text file that determines how the ability looks and what tiles it targets. More on this is in the ShapeParser section.

When the user presses the Ability button, a list of buttons is dynamically created based on the active character class and level using the CreateAbilityList script.

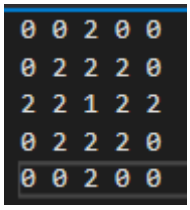


Shape Parser

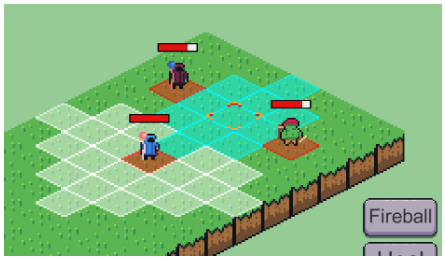
The shape parser takes in a text file and translates that into a list of tiles the ability will hit. The text files contain numbers that determine their shape.

- Number 0 is an ignored tile that will not be targeted by the ability.
- Number 1 is the origin of the ability.
- Number 2 is a tile to be included in the ability.
- Number 3 represents a line of tiles. Wherever there is a 3, the ShapeParser will get all the tiles in the direction of that tile.

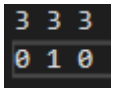
Fireball Shape



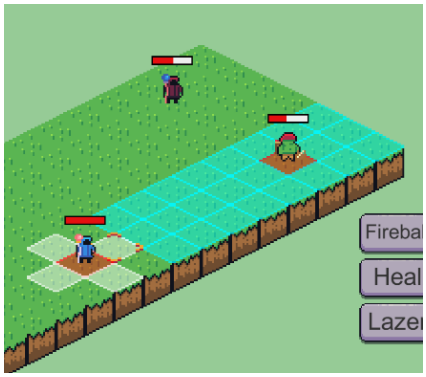
Fireball Ability in Game



Lazer Ability Shape

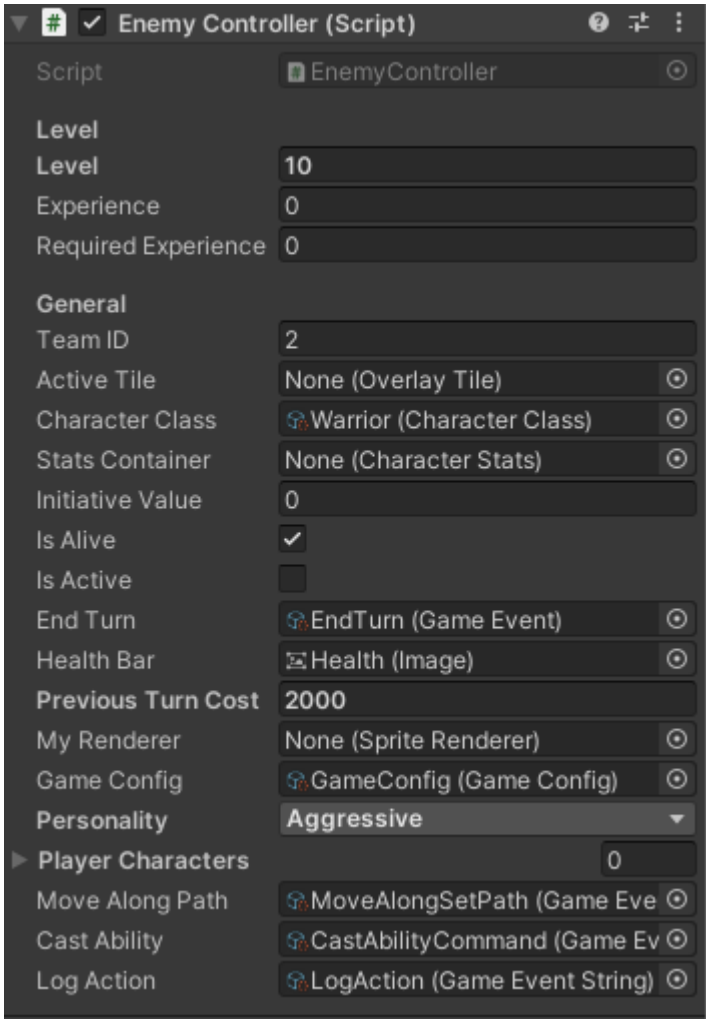


Lazer Ability in Game



Enemy AI

A GDC talk about Xcom 2 inspires the Enemy AI. It uses a scenario system. When it's the Enemy's turn, it will first take all the tiles within its movement range and calculate the best possible scenario, primarily based on the maximum amount of damage it can do based on the primary attack and available abilities. This can be expanded upon an infinite amount based on the game you wish to make. For example, games usually consider the character's positioning, while I don't for now. I intentionally chose this, as you should craft the Enemy AI to fit your game's unique design. But I hope this is at least a good starting point for you.



Turn-Based Mechanics

The turn-based mechanics are the most in-progress work of all the features. The two available turn order options work, but the turn order preview needs more work and will most likely be the first update given to the asset pack unless something else is requested. Though as I said, there are two ways you can calculate turns.

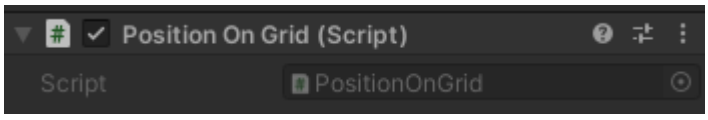
The TurnBasedController contains a dropdown with two available options. Constant and CTB.

1. Constant turn order is pretty simple. When the scene starts, create a list of all the entities and order them by speed from fastest to slowest. Then as you play the game, it will iterate through that list when a character ends its turn.

2. CTB is inspired by Final Fantasy X's Conditional Turn-Based Battle System. Your place in the turn order depends on your actions during the previous turn. Every action as an Initiative cost is added to a character's Initiative Value divided by their speed stat value. I feel the implementation of this may need more work. The current implementation seems quite sensitive in that a slight change to the game's numbers can have a drastic difference. The initiative costs of every action are currently hardcoded in a static Constants file stored in Scripts->Utils->Constants.

Position Entity on Tile

For positioning an object on the tilemap there is a script called Position on Grid that will link it on the closest tile to it's world location.



Final Note

Thank you so much for purchasing this asset pack. I am a single developer learning to make SRPGs in my spare time, and I am incredibly grateful to everyone who has supported me. My ultimate hope for this asset pack is to give developers starting a good head start and get all those prototypes out as fast as possible. I fully intend to continue supporting this pack throughout my development journey, although it may not answer every problem you might have.

Contact Me

If you have any Feedback, Suggestions, or Issues with the project, please do not hesitate to contact me at lawlessplays@gmail.com or my discord, Lawless#7060. I also have a youtube channel where I post updates on the project's progress.