Tactics Toolkit: Pathfinding

Tactics Toolkit: Pathfinding is the final product of a three-part tutorial series where you can see me create this scene from scratch.

Grid-Based Movement With Pathfinding Tutorial: Part 1 - Dynamic Grid Generation

Grid-Based Movement With Pathfinding Tutorial: Part 2 - A* Pathfinding

Grid-Based Movement With Pathfinding Tutorial: Part 3 - Range Finding and Path Display

Tactics Toolkit: Pathfinding also serves as a free sample for my fully featured SRPG Template, **Tactics Toolkit.** Within this template, you'll find a fully functional example of using A* Pathfinding and a Path Preview Display on a Unity Isometric Tilemap.

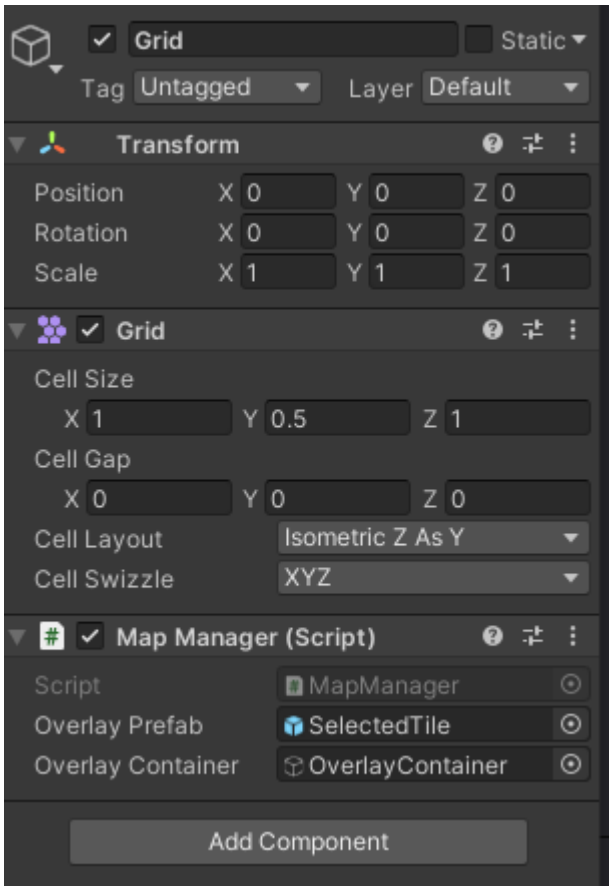
I designed this template with simplicity and education in mind. Within the project, you'll find three folders: **Part 1 - Grid Setup, Part 2 - Grid-based Pathfinding and Part 3 - RangeFinding and Path Display.** Each part is built from the last with some additional complexity so as not to overload anyone. But feel free to go straight to part 3 if you want to see the final product. There is nothing in Part 1 or 2 that does not exist in Part 3.

How to go forward
You can view this project as the starting point for making your own SRPG game. At the moment, all movement and controls are contained within the MouseController. I would recommend starting on the character systems for your game, taking the code from the MouseController and using your character systems instead. For example, this character's movement range is a variable you set via the MouseController. In a more realised game, this would be controlled using a character's class stat as an example.

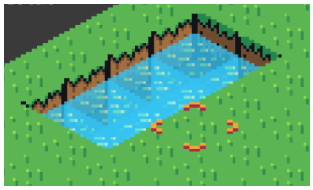
Next, look into turn-based options to get multiple characters on screen. Again, there are many ways to do this, so it's really at the discretion of your game's design that will decide how you go forward. For example, I've implemented a Conditional Turn-Based System in my project. Something reminiscent of the final fantasy series. But perhaps a rigid set order using a characters class stat would be the best place to start. Try multiple approaches to determine how your turn system will feel.

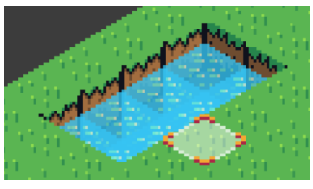
Map And OverlayTiles

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.



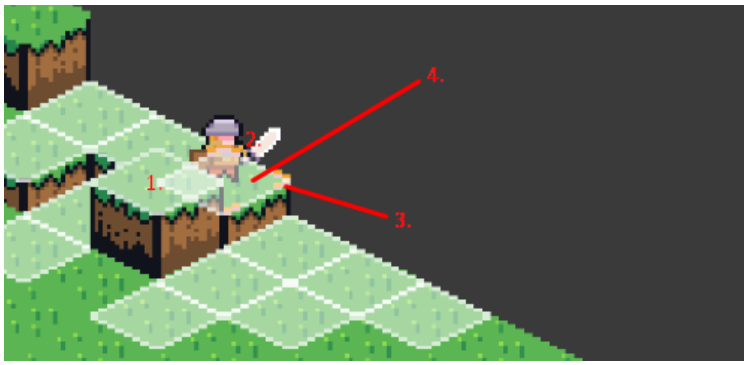
By default, the image of an overlay tile is invisible. But when you interact with the map it is colored to while.



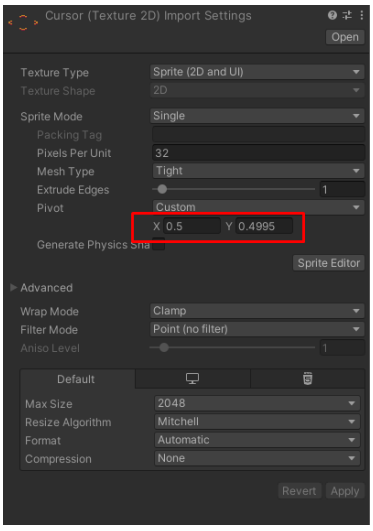


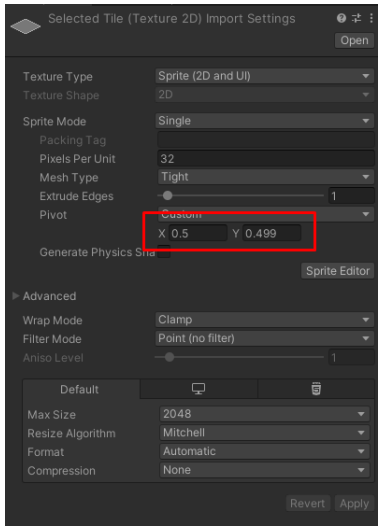
Sprite Layering.

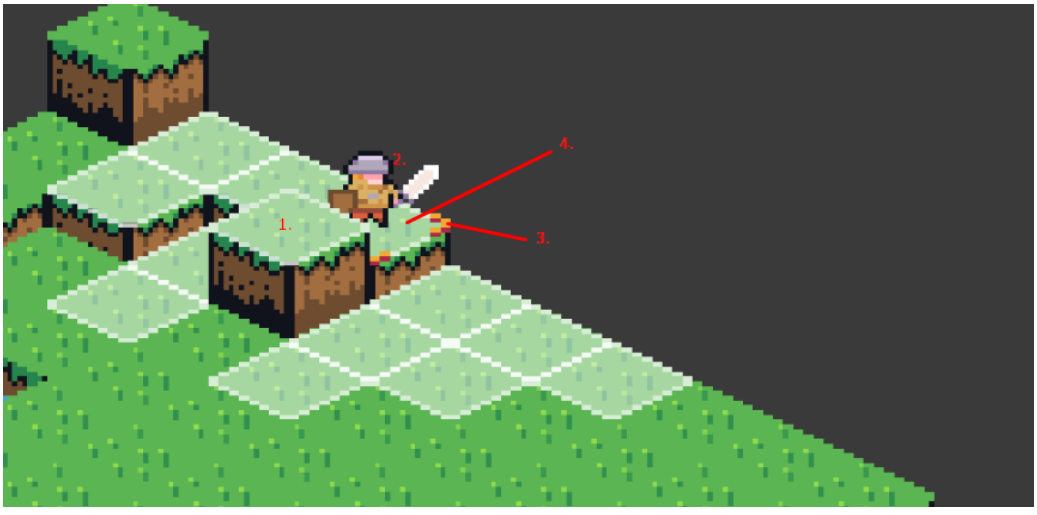
By default Unity will layer images based on the z axis. However is certain instances that there can be multiple image on the same z axis and the rendering order can be incorrect.



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

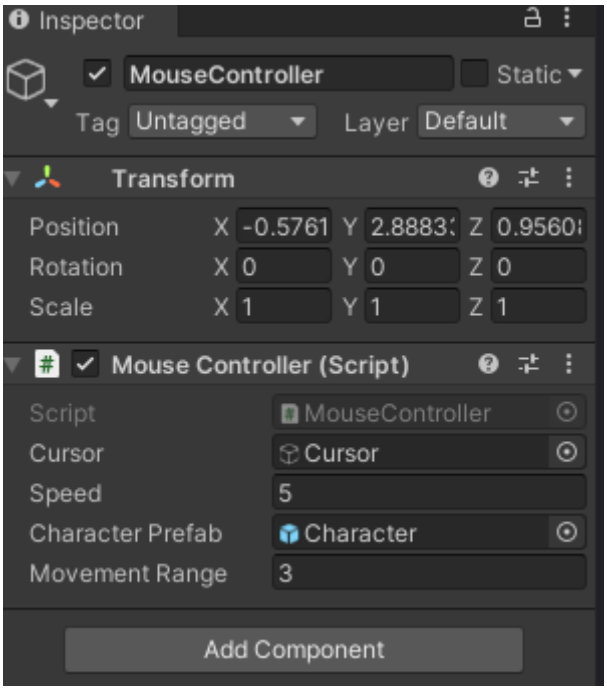






Mouse Controller

The mouse controller handle all the character movement. It contains a reference both the PathFinder and RangeFinder classes. In Part 3, click on the map to spawn in a character. Next the MouseController will get all the inRange tiles using the RangeFinder based on an integer you can set. When you move the mouse within those tiles, the Pathfinder will use A* to get the shortest possible path. Then finally that path will be given to the ArrowTranslator which will display the path to the user.



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.

Final Note

Thank you so much for taking the time to check out 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. In case you haven't scene, this is only part one of a much larger pack that will give you a fully functional prototype for an SRPG game.

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.