Maintenance Guide

Contents

Installation

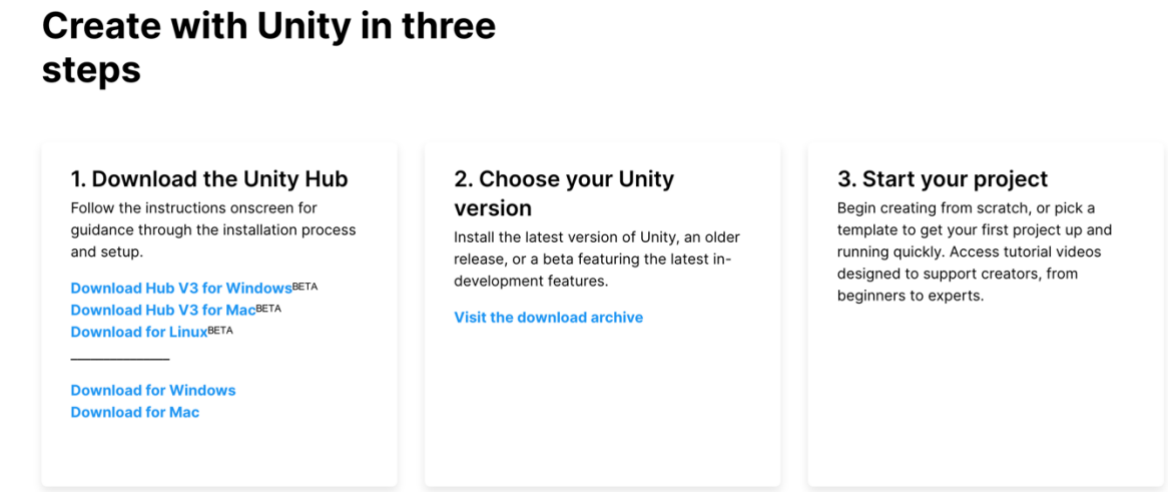
Technical Specifications

Game engine: Unity 2021.2.18f1
Language: C#

How to setup Unity

1. Go to https://unity.com/download

2. Download the software(s) relevant to your device i.e., MacOS, Windows, following the instructions given, as seen below



How to setup the project

1. Clone the repo using terminal using the command ‘git clone or via GitHub Desktop into a location you have access to. The folder will be named ‘AutoDungeon’

2. Open up Unity Hub if installed, if not refer to ‘How to Setup Unity’

3. Click the ‘Open’ button in Unity Hub (top right corner)

4. Locate the cloned repo on your device (a pop up will appear giving you access to the folders on your device)

5. Click ‘Open’ after selecting the correct folder and the project will open in Unity and Unity Hub will provide a shortcut to it from that point forwards.

6. Move the ‘Map 1’ Scene into the hierarchy from the assets folder and make sure it is loaded by selecting the 3 dots to the far right and selecting ‘Load Scene’ if the option is available. If the ‘Load Scene’ option is not available, the scene is already loaded.

Game Logic

1. Map 1 scene is loaded on startup

2. The player's mouse cursor is replaced with a sprite. The sprite changes when hovering over an interactable node.

3. Interactable nodes are indicated by a glow. Greyed out nodes indicate they have already been selected or are on a path which can no longer be traversed

4. Possible paths are indicated in gold. Greyed out connectors indicate paths that have already been traversed or are on a path which can no longer be traversed

5. The types of nodes are the following:

    a. If node is a chest, a pop-up will appear with a random number of units between 1 and 6
    b. If node is is a skull alone, the scene will change to the grid scene which will have easier units
    c. If node is a skull and crossbones, the scene will change to the grid scene which will have harder units
    d. If node is a key, a pop-up indicates a new region of the map is unlocked
    e. If nose is a star, the level is a popup indicates you've beaten the game

6. At some nodes the player must choose between branching paths. If a path is selected, the other will no longer be accessible and will be greyed out.

7. After clicking on a node, the next node(s) will become interactable

8. The levels are a grid playing area with a banner to the left-hand side of the screen providing the user with the number and type of units they have, and a button labelled ‘fight’ to the right

9. Users must drag the unit onto a tile on the grid of their choice. The user does not have to use all the units they have To drag, click on the button next to the unit of choice, then drag the cursor to the tile of choice, and then click again to drop.

    a. Melee unit: deals 2 damage to targets within a 1 tile range, has 5 health.
    b. Ranged unit: deals 6 damage to targets within a 5-tile range, has 4 health, has half the attack speed of the melee unit
    c. Easy enemy unit: deals 1 damage to targets within a 1 tile range, has 5 health*
    d. Hard enemy unit: deals 4 damage to targets within a 1 tile range, has 14 health*

    *Enemy units gain an additional health after every successful battle

10. Click the fight button to begin the game, and wait for a win or lose pop-up to appear

    a. If lost, the pop-up gives the option to try again
    b. If won, the option is to return to the map

Maintenance

This section includes items that have been coded/scripted. The code can be modified and updated in an IDE of choice that supports C#. It also includes all the technical items of the game that have not been coded/scripted. These assets will have to be maintained and updated within Unity.

The technical assets can all be found in the Assets folder of the Unity project stored in folders named Sprites, Prefabs, Scripts, Scenes, and Tiles. Some scripts may have overlapping functionalities for different game elements.

The game is built with a combination of scripts and game objects called prefabs in Unity. You can add a script to a game object by clicking on the game object which will open up the Inspector window to the right of the screen. In the Inspector window, click 'Add Component' to attach a script or a different game object to the current one. The background for the map and the grid are built using Tilemap.

Scenes

The game is split across 2 scenes currently which the player switches between throughout the game.

Map 1

Loads the map and related elements. This includes the tiles background, camera, nodes, connectors, lights, UI, and cursor.

Grid

Contains all the objects that deal with the combat part of the game e.g. getting up the enemy team formations and building the grid graph

Game Objects

Game object	Sprite	Script	In scene	Purpose
Main Camera	None	DragCamera2D.cs CameraPosition.cs CameraBounds.cs	Both	Allows drag to move camera, zoom, and changing the camera position when reloading the scene
UIManager	None	UIController.cs	Both	Provides methods to create and display PopUps with custom messages. Called via CreatePopup method.
GameManager	None	GameManager.cs	Both	Stores information required across both scenes; unit pool, difficulty of selected encounter, and total number of battles won. Also holds currently redundant methods for instantiating units on grid. Responsibility for this now assigned to unit manager.
Canvas	None	UnitPoolDisplay.cs	Both	The canvas is the object on which all UI elements are rendered. Ensures appropriate relative

				placements and scaling with different resolutions/aspect ratios.
EventSystem	None	None	Both	Standard Unity object, whenever a Canvas is created an EventSystem is created to listen for and pass on events such as mouseClicks or mouse hovers.
cursor	Cursor(1) and Cursor(2) from Fantasy Cursor Pack	MouseCursor.cs	Map 1	Responsible for context dependant cursor swapping. In this scene, responsible for changing cursor to unit currently selected for placement.
leveltemplate	Nodes use Tiny Fantasy Icons: Bones_B, Bones_C, ChestA, Key_B, Star_B Labels use Black Family from font-generator.com Grid uses Terrain Tile Hex Samples	ObjectClick.cs ConnectorInfo.cs NodeInfo.cs	Map 1	Display the tiles that form the map background and tracks the player's progression through the nodes and connectors on the map
CameraBounds	None	CameraBounds.cs	Map 1	Bounds camera so player does not see outside map
Grid	Tileset used ‘Pixal Art Top Down – Basic’	GridGraph.cs HoverHighlight.cs	Grid	Holds the design of the grid. 'Tilemap_grid' layer determines the size of the grid. Other layers are purely for cosmetic proposes.
GridManager	None	GridManager.cs	Grid	Instantiate new graph, create node and connect them for each tile. Manage the coordinates of the grid and hold the nodes’ positions.
UnitManager	None	BaseUnit.cs UnitManager.cs	Grid	Handle win/lose conditions. Create instances of units.
Return arrow	back-arrow-icon	returnMap.cs	Grid	Forces a scene change from Grid to Map
Lights	None	GridLight.cs	Grid	Illuminates the Grid scene differently depending on active node number

Main Camera

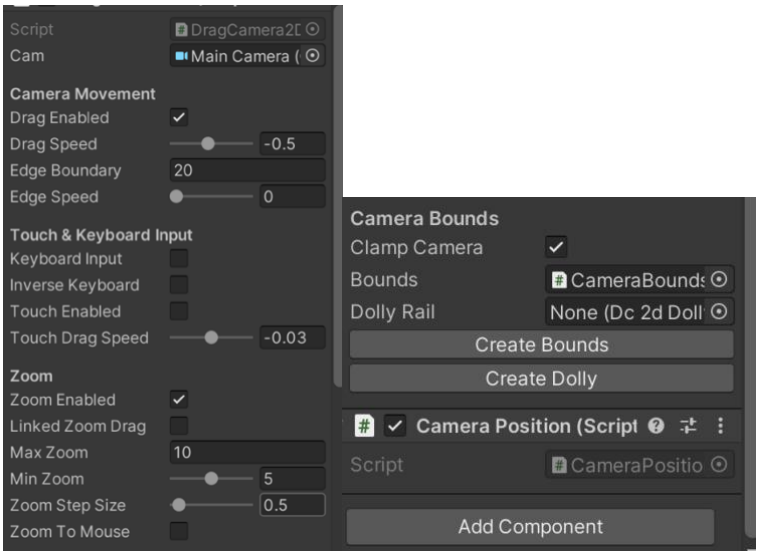
1. Determines the area of the game scene (and therefore the objects) displayed to the user when played.

2. The camera uses a 2D unity asset that provide many useful functions. The functions we use are pan, zoom, and bound camera.

3. The DragCamera2D script is added to the main camera. In the Cam field, the Main Camera has been dragged in. The inspector panel provide a user-friendly interface to change the camera settings, which we have used rather than editing the code.

4. We enable clamp camera and click create bounds which creates the camera bounds object which we configure so that the player cannot pan the camera outside the map.

5. There is also a user guide in the documentation folder for this asset.



6. When the map scene is reoladed the camera's position changes depending on the active node (see ObjectClick script). The range of active nodes correspond to the 'biome' the player is currently in and since the map is grid based, the camera only moves 9 units up or down or 15 units right. This is necessary as it prevents the camera resetting to (0,0) after reloading the scene after battle.

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class CameraPosition : MonoBehaviour
{
    void Start()
    {
        if (ObjectClick.ActiveNode <= 4)
        {
            transform.position = new Vector3(0, 0, -10);
        }

        if (5 <= ObjectClick.ActiveNode && ObjectClick.ActiveNode <= 10)
        {
            transform.position = new Vector3(0, 9, -10);
        }

        if (11 <= ObjectClick.ActiveNode && ObjectClick.ActiveNode <= 16)
        {
            transform.position = new Vector3(0, 18, -10);
        }

        if (17 <= ObjectClick.ActiveNode && ObjectClick.ActiveNode <= 22)
        {
            transform.position = new Vector3(15, 18, -10);
        }

        if (23 <= ObjectClick.ActiveNode && ObjectClick.ActiveNode <= 28)
        {
            transform.position = new Vector3(15, 9, -10);
        }

        if (29 <= ObjectClick.ActiveNode && ObjectClick.ActiveNode <= 34)
        {
            transform.position = new Vector3(15, 0, -10);
        }
```

7. Each scene has its own instance of the main camera. The camera in the grid scene only displays the grid scene and does not need any additional functionality.

8. The main camera is referenced directly by both canvases, as they are drawn only onto the area currently defined by the main camera.

UIManager

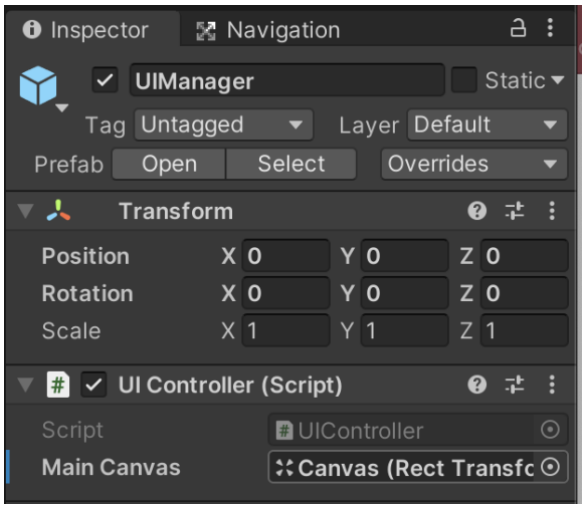
1. Located in the prefab folder as 'UIManager'

2. Used in both the Grid scene and the Map scene

3. Connected to the UI Controller script (script can be found in the scripts folder)

4. UI Controller script creates the popups that the user can interact with including the Level Win and Level Lose pop-ups

5. Works with the Pop Up script via UI Controller



How the UI Manager should appear in inspector

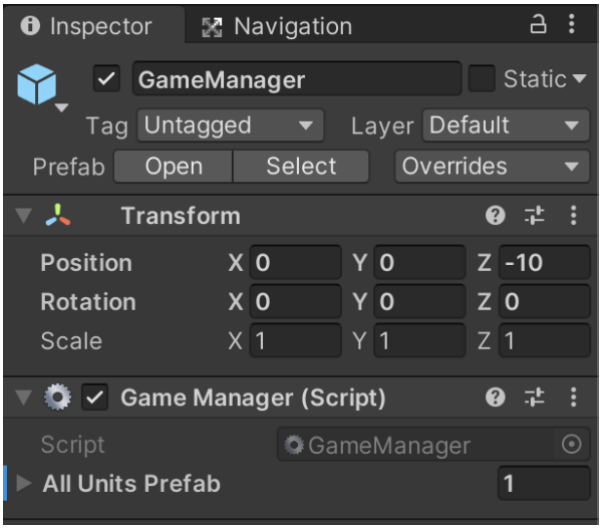
GameManager

1. Located in the prefabs folder as ‘GameManager’

2. Used in the Map scene as a game object

3. Starts up the game and acts as an anchor for the Base Unit script to the map i.e., allows unit instantiation and generates pseudo-random number for formation generation

4. The loader script instantiates the Game Manager

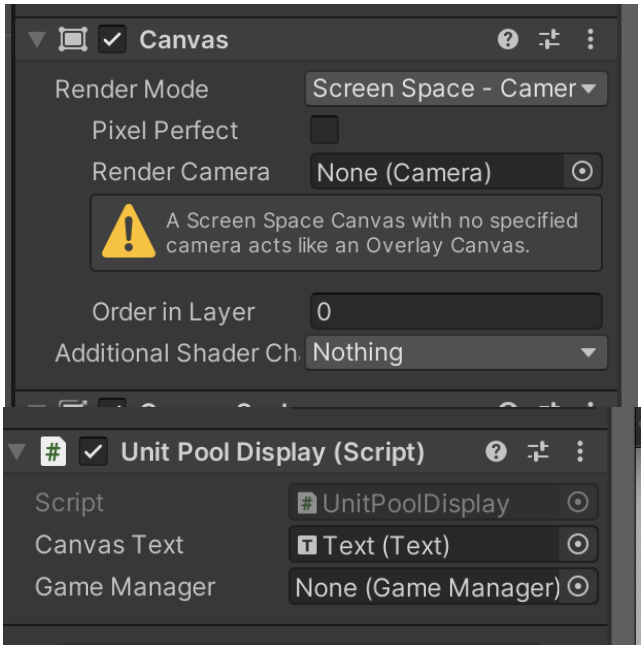


How the Game Manager looks like in Inspector

Canvas

1. In the Map 1 scene, the canvas is:

    a. Connected to the Unit Pool Display script
    b. Currently only displays number of units player has.
    c. Can be improved to provide a button to take user to a settings menu and a return to main menu button (inside script)
    d. Can be improved to display other user stats such as user gold, specify the count of the different types of units the user has, etc. (inside script)

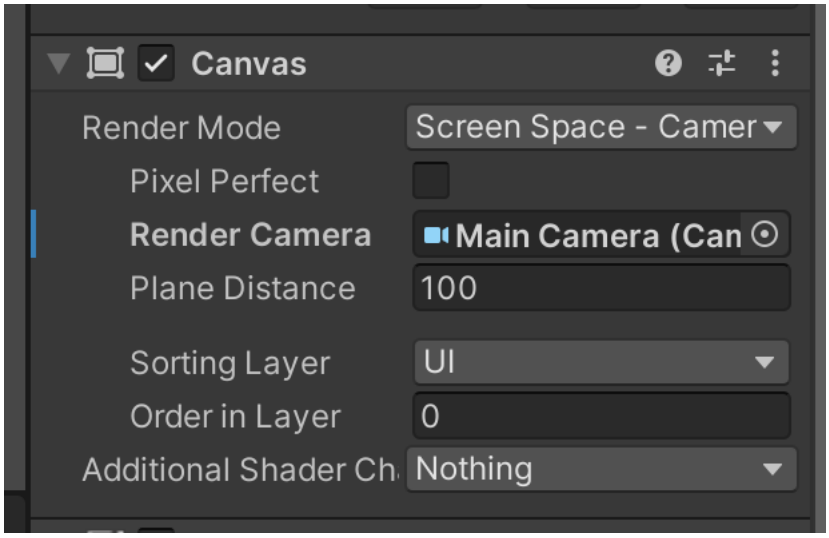


In the Inspector, make sure the Render Camera is not set to anything and the unit pool display is attached to the canvas prefab. There is a possibility of including the Game Manager if the functionality of the display is increased.

2. In the Grid scene, the canvas:

    a. Contains the Unit Selection game object and script, as well as the Start Fight game object
    b. Currently only displays units available and provides the button for the drag and drop.

c. Can be improved to add in a kill feed, add user-controlled actions like use potions, display number of units remaining on battlefield, pause game button which brings up menu with option to restart, resume, or return to map, settings button, and toggle sounds button.



In the Inspector, make sure the Render Camera is set to Main Camera like above

EventSystem

1. Standard Unity Object created whenever a Canvas is added to the scene. Listens for 'events' in the scene (often collisions, mouse clicks etc), and passes on these events to game objects which are 'subscribed' to these events.

2. Please review C# documentation for more detailed description of C# events if required.

cursor

1. Used inside the 'Map 1' scene (labelled as 'cursor')

2. Prefab can be found in the prefabs folder under Assets

3. Connected to the Mouse Cursor script and Object Click script through the level template object, both found in the scripts folder.

4. Changes cursor sprite when hovering over interactable game object and provides click event when interacting with certain game objects i.e., level nodes on the map to load the level, etc. This is identified through a 2D raycast

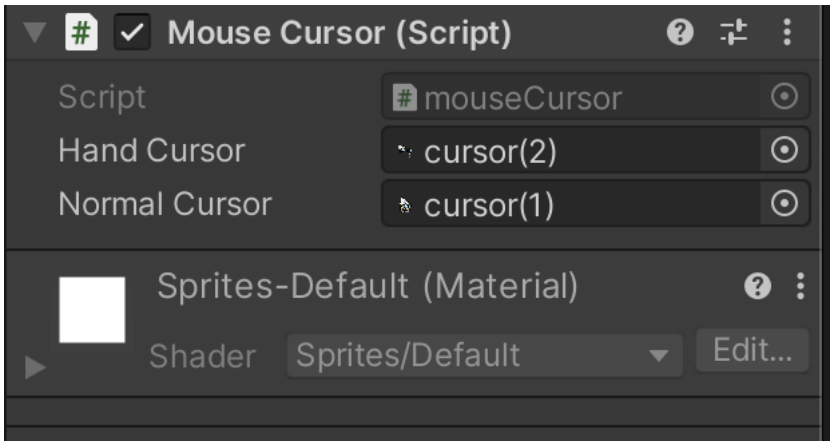
```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class MouseCursor : MonoBehaviour
{
    public Texture2D handCursor;
    public Texture2D normalCursor;
    private Vector2 normalOffset = new Vector2(24, 16);
    private Vector2 handOffset = new Vector2(18, 20);

    void Start()
    {
        Cursor.SetCursor(normalCursor, normalOffset, CursorMode.Auto);
    }

    void Update()
    {
        // Hand if hover over interactable sprite
        Vector2 cursorPos = Camera.main.ScreenToWorldPoint(Input.mousePosition);
        transform.position = cursorPos;

        RaycastHit2D hit = Physics2D.Raycast(cursorPos, Vector2.zero);
        Cursor.SetCursor(normalCursor, normalOffset, CursorMode.Auto);

        if (hit.collider != null)
        {
            updateObject(hit.collider.gameObject);
        }
    }

    void updateObject(GameObject go)
    {
        if (ObjectClick.ActiveNode == go.GetComponent<NodeInfo>().NodeNumber)
        {
            Cursor.SetCursor(handCursor, handOffset, CursorMode.Auto);
        }
    }
}
```

5. The mouse also appears in the Grid under the Canvas object as Unit selection

6. Also appears as the highlight during the level that lets user know what tile they are hovering on, through the Grid object in the Grid scene

7. Script for the highlight is called HoverHighlight.cs and can be found in the Scripts folder

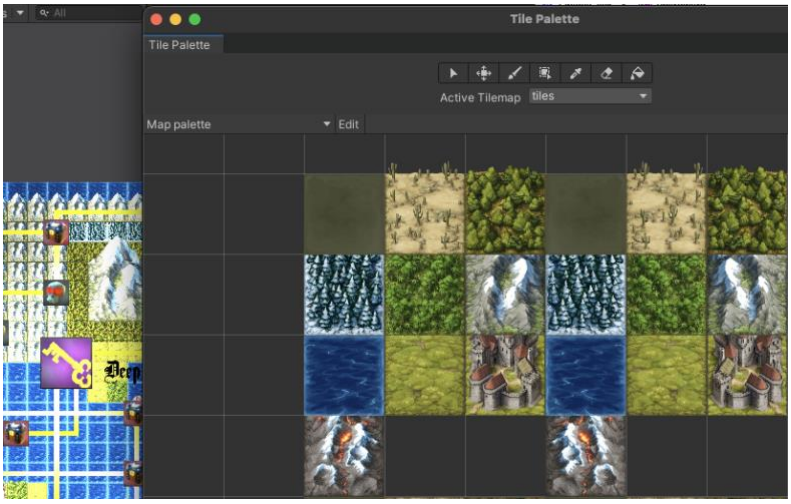
8. The Unit Selection is connected to the Unit Selector script and enables the user to drag and drop units onto the Grid by clicking the relevant button for the unit, then moving to the tile the user wishes the unit to be on, and then clicking on that tile



How the Mouse Cursor Script Appears in the Cursor Prefab in Inspector

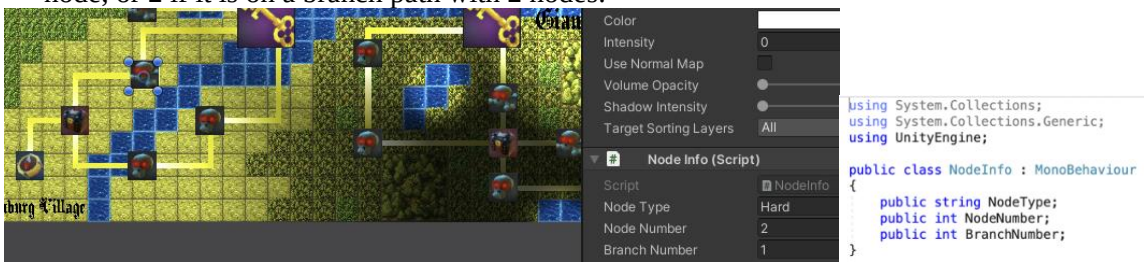
leveltemplate

1. The map is built using Unity's Tilemap. Tiles can be easily edited by clicking the tiles object then open tile palette. It is possible to create many additional maps using the map palette we have created and the self-explanatory tools at the top of the tile palette window.



2. All nodes are assigned a simple script that gives them additional public properties

    a. Node type is a string that affects what happens when the player clicks the node. Current options are 'Start', 'Treasure', 'Hard', 'Easy', 'Key', and 'End'.
    b. Node number is the order of the node in the map. It ascends by 1 for each unique set of nodes.
    c. Branch number is either 0 if it is not on a branching path, 1 if it on a branching path with 1 node, or 2 if it is on a branch path with 2 nodes.



This has a skull and crossbones so is 'Hard'
It is the second node on the map (counting from 0) so is node number 2
It is on the path with 1 node so is on branch number 1
The node directly below this one is also node number 2 but is on branch 2
The node to the right of this is node number 3 on branch 2
The key is node number 4 on branch 0

3. All connectors are assigned a simple script that gives them additional public properties

    a. Connector number is the order of the connector in the map. It ascends by 1 for each unique start node it comes out from
    b. Connector sub number is either 0 if the end node is unique, 1 if the end node is not unique and it's on the path with 1 node, or 2 if the end node is not unique and it's on the path with 2 nodes

4. The objectclick script so far only supports the node types listed above and either a linear path or a branching path where there is 1 node on one path and 2 nodes on the other path. Nodes and connectors need to be labelled carefully to prevent unwanted behaviour.

5. The script has 3 parts:

    c. First is the code that runs when the scene starts i.e. the game starts or the player returns from battle. It loops through all the nodes and connectors to find where the player is now on the map (the active node) and turns the appropriate elements grey
    d. Second is the code that runs when the player clicks on the next interactable node and how that affects the map i.e. which elements turn grey, which node now illuminates
    e. Third is the code that runs depending on the type of node clicked

6. The script could be adapated easily to add additional node types by changing the last part of the code. It is more difficult to add alternative path layouts.

CameraBounds

1. Holds a point variable which define the dimensions of the area encompassed by the main camera.

2. This is currently stored as a vector 3 object, with length of 39, height of 22. Increasing or decreasing these values within the inspector will increase or decrease the area bounded by the main camera.

Grid

1. Located in the Grid scene as ‘Grid’

2. Contains the tilemap design for the grid

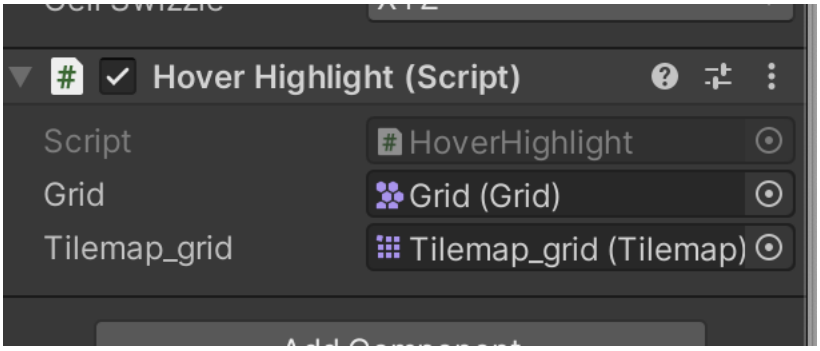
3. ‘tilemap_grid’ is passed into ‘GridManager’ to instanciate a graph for the grid

4. Other layers of tilemap are purely cosmetic

5. ‘HoverHighlight.cs’ was attached to this asset so that when player select a certain tile, the tile will change the color.

6. ‘gridgraph.cs’ also contain the pathfinding method, which can find the shortest path between two nodes on the grid. It traverses from the start node to the end node, updates the path until it finds the shortest path. Finally, it returns the path.

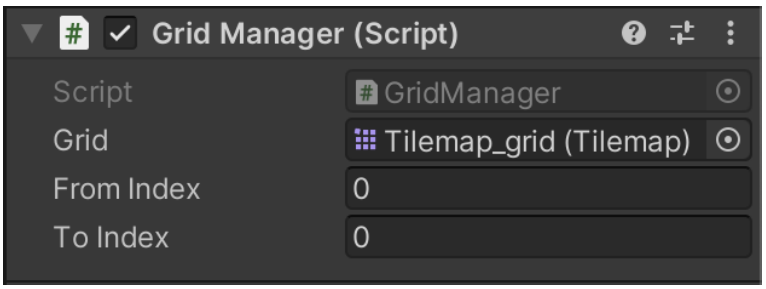
7. Helps with pathfinding for units by providing access to the graph



How the hover highlight is added to the Grid game object in Inspector

GridManager

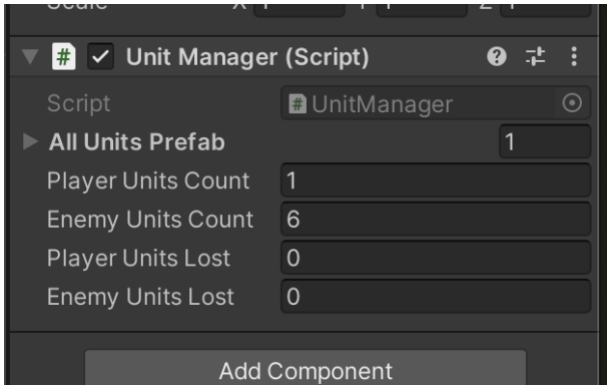
1. Located in the Grid scene as 'GridManager'
2. Takes in the 'tilemap_grid' layer of the tilemap, create a node and store its' position for each tile and connect them together to form a graph
3. Select a formation randomly for the enemy team and place them on the grid when the battle starts
4. For each unit on the grid, find the shortest path to the closest opposing unit
5. In scene view, display dots for each node and lines for edges between neighbouring nodes for debugging proposes



How the Grid Manager script is added to the Grid Manager object in the inspector

UnitManager

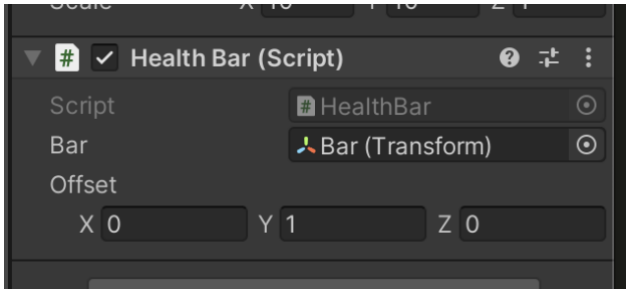
1. Located in the Grid scene as 'UnitManager'
2. The Unit Manager creates the units with set defaults and monitors unit numbers
3. Unit Manager also determines the formation and positions of the enemy units
4. Unit Manager acts as the Game Manager during levels



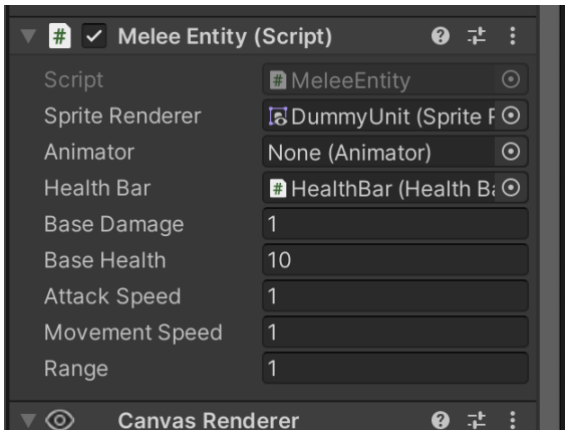
How the Unit Manager appears in the Inspector:

Base Unit

1. Located in the prefab folder under 'DummyUnit'
2. Located in the scripts folder as BaseUnit.cs
3. Located in the Grid scene as a prefab called dummy unit
4. There is also a Unit Manager that works together with the Base Unit to control the two different teams (player team and enemy team)
5. Connected to Melee Entity script, Health Bar prefab, the Base Unit script through the Melee script, the Unit Manager script, and the Unit Manager prefab via the Unit Manager script
6. Contains information on the unit's base health, damage, attack speed, movement speed, and range
7. Tracks unit's current target, team affiliation, state (dead or not), current location, next location, and whether it's moving or not
8. Gives the unit the ability to find a target, damage a target, move to a target, and take damage
9. All unit types currently existing (ranged and melee) use the dummy unit default with the Game Manager providing different stats for range, health, damage etc
10. Was meant to be a default and interact with scripts like Melee Entity to provide different kinds of units but a workaround was developed that made the idea behind the melee entity script useless. If the different kinds of units increase, it would make sense to revert back to this idea of having different scripts for each unit type.
11. The Health Bar 'floats' above the unit and provides a HUD of the unit's current health
12. Health bar script tracks the position of the unit to maintain its position and scales the Health Bar based on the health remaining
13. Potential to add Mana bar or Ultimate timer to provide units with other kinds of attacks rather than physical with set amount of damage
14. Health Bar prefab is directly attached to the dummy unit prefab.
15. Health Bar prefab also has two objects attached to it, the background and the bar.
16. The bar is scaled up and down depending on the unit health
17. Both use a basic white 10 pixels x 1 pixel image for a sprite (bar sprite is coloured red)
18. The offsets for the Health Bar prefab keep it centred above the unit's head



How the Health Bar script is connected to the Health Bar prefab in Inspector



How the Melee Entity script and Health Bar is connected to the dummy unit prefab in Inspector

Return arrow

1. The return arrow is mainly for testing purposes as it allows you to return to the map scene from the battle scene without having to complete the battle

2. It uses a simple change scene script when clicked which is identified by a 2D raycast as the object has a 2D collider

3. This asset could be repositioned so it is hidden from the player's camera in the playable version of the game

```
// Update is called once per frame
void Update()
{
    if (Input.GetMouseButtonDown(0))
    {
        Vector2 rayCastPosition = Camera.main.ScreenToWorldPoint(Input.mousePosition);
        RaycastHit2D hit = Physics2D.Raycast(rayCastPosition, Vector2.zero);

        if (hit.collider != null)
        {
            SceneManager.LoadScene("Map 1");
        }
    }
}
```

Lights

1. The Grid Light script adds cosmetic changes to the levels making them look appropriate for the realm they are supposed to be in e.g., a green tint for forest levels

2. The code is very similar to the camera position script in that depending on whether the active node is in range, the light in the battle changes

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.Experimental.Rendering.Universal;

public class GridLight : MonoBehaviour
{
    void Start()
    {
        if (ObjectClick.ActiveNode <= 4)
        {
            GetComponent<Light2D>().color = Color.white;
        }

        if (5 <= ObjectClick.ActiveNode && ObjectClick.ActiveNode <= 10)
        {
            GetComponent<Light2D>().color = Color.green;
        }

        if (11 <= ObjectClick.ActiveNode && ObjectClick.ActiveNode <= 16)
        {
            GetComponent<Light2D>().color = Color.blue;
        }

        if (17 <= ObjectClick.ActiveNode && ObjectClick.ActiveNode <= 22)
        {
            GetComponent<Light2D>().color = Color.cyan;
        }
```

3. Additional 2D lights can easily be created in the gameobject window by right clicking, selecting light and then 2D

Extending the game

Main menu

The main menu could be made up of:

- Start button
- Settings button
- Exit game button

The start button should either take the user directly to the map or it could be developed further. The further development means it would take the user to a new screen where they can either load a previous save or start a new game and from there move on to the map.

The settings button would take the user to a new screen where they can adjust the game to their needs. The settings screen would also need a button to return to the main menu. Potential settings include:

- Volume adjustments
- Brightness adjustments
- Screen size adjustments
- Screen quality adjustments
- Accessibility options

Exit button closes the game and returns the user back to their desktop/phone screen. The exit button will not be necessary if the game remains as a browser game.

Sound

The game is currently without sound. Potential sounds to make include:

- Theme Song/Main menu song
- Map BGM
- Appropriate Level BGM
- Button sounds
- Player and enemy sounds:
    - Death sounds
    - Attack sounds
- End Credits song

More units

A larger variety of units can add a lot more depth to the combat and strategizing of the game:

- Melee shield units with a large health pool but slow and weak attack
- Classic ‘glass cannon’: ranged magic casting units that deals huge damage but is low on health
- Area-of-effect attacking units that can attack several tiles at one time
- Healers that heal friendly damaged units but cannot attack

More enemies

Enemy types can be developed that are more appropriate to the location they are in. For example, when in water levels, the enemies faced can use water-based attacks, travel around the grid through water puddles represented by blue tiles and have appropriate looking sprites.

Boss level enemies could be included to provide more challenge and the opportunity for better rewards. They could also signal the end of an area as well. A last boss is an option for the last level and could reflect all the different enemies the player has encountered throughout the game.

Acknowledgements

1. Fantasy Wooden GUI – Unity Assets Store – blackhammer.co@gmail.com - Used under standard Unity Asset Store EULA Fantasy Wooden GUI : Free | 2D GUI | Unity Asset Store

2. Tiles and Hexes: 2D Painted Terrain Samples - Unity Assets Store - dgbaumgart@gmail.com - Used under standard Unity Asset Store EULA https://assetstore.unity.com/packages/2d/environments/tiles-and-hexes-2d-painted-terrain-samples-191945

3. Tiny Fantasy Icons - Unity Assets Store - vespawarrior3d@gmail.com - Used under standard Unity Asset Store EULA https://assetstore.unity.com/packages/2d/gui/icons/tiny-fantasy-icons-99722

4. Drag Camera 2D - Unity Assets Store - darkdragon_010@hotmail.com - Used under standard Unity Asset Store EULA https://assetstore.unity.com/packages/tools/camera/drag-camera-2d-126070

5. Pixel Art Top Down – Basic – Unity Assets Store – Used under Standard Unity Asset Store EULA https://assetstore.unity.com/packages/2d/environments/pixel-art-top-down-basic-187605

6. Fantasy RPG Cursor Pack – Unity Assets Store - denoprimo@yandex.ru - Used under standard Unity Asset Store EULA Fantasy RPG Cursor Pack | 2D Icons | Unity Asset Store

7. Free 2D Archers Sprites – Unity Assets Store - office@honeti.com - Used under standard Unity Asset Store EULA FREE 2D Archers Sprites | 2D Characters | Unity Asset Store

8. 2D Skeletons Sprites – Unity Assets Store - office@honeti.com - Used under standard Unity Asset Store EULA 2D Skeletons Sprites | 2D Characters | Unity Asset Store

9. https://www.font-generator.com/fonts/BlackFamily/