# GPROG Artefact Planning

Primary mechanic -> multi-layer Astar pathfinding

Secondary mechanic -> inventory/held item system/puzzle mechanics

Overview:

My game will be a puzzle game, involving a 2d isometric grid. The player clicks on a tile to move to it, can pick up ‘puzzle pieces’ and carry one at a time. The player can place puzzle pieces on podiums to interact with the environment and collect more puzzle pieces. Once the player has collected all the puzzle pieces for a level, they can progress to the next level until they beat all the levels/

Task List:

* Pathfinding
  + The player will move by left clicking on a visible tile, an A-star pathfinding algorithm will then calculate a path to that tile. The player will then walk across that path until they have reached the destination tile
  + The grid will be isometric allowing for three dimensions to be displayed on a 2d screen, the pathfinding algorithm must be able to locate ‘layer-traversal tiles’ such as stairs or ramps so the the player can move between the layers
  + Extension: If the player clicks on a visible podium, the pathfinding algorithm should calculate the closest adjacent tile. Double extension: without calculating and comparing traversal costs for all adjacent tiles.
* Puzzles
  + Player must be able to pick up puzzle pieces, only one at a time
  + Player must be able to ‘hold’ a puzzle piece and move with it
  + Player must be able to place a puzzle piece on a pedestal
  + Pedestals should open doors or reveal new areas with different pieces in order to create complex logic puzzle and engage the player
  + Once the puzzle is complete – the game finishes/the player progresses to the next level

# Notes during creation:

* I created a multi-layer isometric grid with each layer being a different game object as a child of my grid
* I created a custom tile script which inherits from the base unity one and is a scriptable object in the asset menu
* I began working on the node grid logic – only focusing on setting up pathfinding in two dimensions to begin with
  + Setting up the node with a position and the appropriate pathfinding cost variables
  + Setting up the nodegrid class which will contain a two dimensional array of node objects with the appropriate positions to match the unity tilemap objects
  + Astar class containing multiple functions to calculate paths
  + Player movement script which uses mouse clicks as an input
  + Prototype the pathfinding mechanics by using Debug.Draw line to showcase the paths
* Then work on the player traversing the path + set up walking animations
* PROBLEM: I ran into a bug where if there was lag, the player would continually walk in a certain direction. I resolved this by updating the direction the player should travel in before moving the player in that direction as opposed to only calculating the direction once.
* PROBLEM: pathfinding costs for traversing south east seem to be broken :D – Solution: When calculating the distance cost, I was incorrectly calculating the difference of the y because I was using the x variable for node a….. it is embarrassing how long it took me to find it.
* Then work on making the grid respect three dimensions
  + First I must make the grid three dimensional
    - Changing the node grid to a three dimensional array of nodes, changing the tilemap reference to a list of tilemaps. Then changing calls to Tilemap.gettile() function to involve the z position of the node
  + Then, I must find the highest z position that has a valid tile when the player clicks on a location so that the player can move across layers. I did this by iterating backwards through all the layers of the node grid and checking if there is a valid tile at that position on that layer.
    - PROBLEM: I ran into an issue later on where the tile being selected was in a position north from the position selected of a distance equal to the layer height. I solved this by subtracting the z position from the x and y positions when calculating the selected tile
  + Then I must make the pathfinding algorithm recognise stairs and slabs as a method to traverse between layers.
    - I needed to find adjacent tiles upon multiple layers. To achieve this, I could have written a new function or ran the function twice with two different lists as outputs, but instead I chose to give a reference to a list of nodes
    - PROBLEM: when the player changes layer, the GetCellCenterWorld function seems to be offset north by half a tile. I realised that this was because the GetCellCenterWorld function offsets by the z position, however I wasn’t using this so when calling this function I set the z position to 0
  + I realised that the player was able to walk through walls as my pathfinding algorithm wasn’t checking if there were any tiles above the tile when checking its validity so I wrote a new function to include all the necessary validity checks. I also made some more functions to improve the readability of my code.
  + My player wasn’t displaying behind tiles that my player should be displaying behind. After some research I discovered I needed to set the Transparency Sort Axis to a custom axis of x:0 y:1 z:0 in the graphics project settings. I also had to update my player’s sprite renderer’s sorting order BEFORE moving up a layer, but AFTER moving down a layer to prevent the sprite from clipping behind the stairs
* I wanted the pathfinding algorithm to pathfind to the closest tile to the player, which is adjacent to a podium when the player clicks on it. However, I wanted to push myself to come up with a solution with optimization in mind. An obvious solution would be to generate paths to all the adjacent tiles and traverse the one with the lowest total cost, however that is the most brute-force method. I then thought perhaps I could calculate each path simultaneously and then stop generating when a path has completed but I wasn’t even sure if this would be a more efficient solution. I then realised the best solution was significantly simpler than I originally thought. All I had to do was generate a path to the tile the podium was on and remove the last node from the path.
* I had the visuals, the grid snapping and the node occupancy set up for my tiles, now I needed to make the podiums hold things. And, allow the player to carry, pick up and place puzzle pieces. This was also when I finally attached the main camera to the player with a y-offset of 1 in order to complement the isometric viewpoint
  + I created a basic orb asset which I can change the colour of in Unity in order to create different puzzle pieces
  + I made this bind to a selected podium on start, I also made it so if the player clicks on a podium, they will walk to it and “interact with it” this means they will pick up the orb if they are not holding anything, or they will place the orb they are holding if the pedestal is empty.
  + I struggled for a while with why the pedestal wasn’t being interacted with when the player clicks on it while already stood next to it. This was a lovely way to discover that async functions do not throw errors or at least it didn’t in this case. The problem was that I was trying to get a node to travel when pathing to the podium next to me, but the path was empty so the index was out of bounds. I fixed this by checking the length of the path list before attempting access.
* Now that the podiums accept and give items, I need to make podiums do something when they hold items
  + I want to do some kind of power lines system inspired by Minecraft redstone where podiums have different coloured wires coming out of them leading to tiles that they interact with. Either opening doors or creating bridges. This means the same podium could do different things when a different coloured orb is placed inside it, opening up more possibilities for puzzlecrafting
  + The way to progress would be a door that opens when the appropriate podiums have the correct orbs in them.
  + I thought about making the wires on another grid, like the one I was using for the tile and pathfinding, but I wanted the wires to light up with a delay when becoming powered, and I thought making the wires into tiles would be too restrictive as tiles are references to the tile object rather than game objects, they are not each unique instances.
  + I made a wire class and a power line class. The podium will contain a list of all the powerlines connected and iterate through them to call any appropriate ones. The powerline script then iterates through all wires that are a child of it and powers them up/down appropriately. The tiles that are powered inherit from the wire class and thus can be called the same way.
  + Some tiles are disabled by default, and some are enabled by default.
* The last thing to do is to create a goal tile, one which progresses the player to the next level
* Now that both my primary and secondary mechanics are implemented, I just need to add some menus, perhaps a level select, create a few levels and polish the game a little.
  + I got some feedback from a few players. I was worried that the grid may be hard to read or that the visuals of the powered tiles may be confusing, but my players seemed to have little to no problems in that regard. The most common negative feedback was that the player should be able to click on the podiums rather than the tile they are on. So, I added colliders to my podiums and checked if the player clicks on them before checking what tile the player clicked on.