The goal is to have a playable version of Realm Rush by November 30

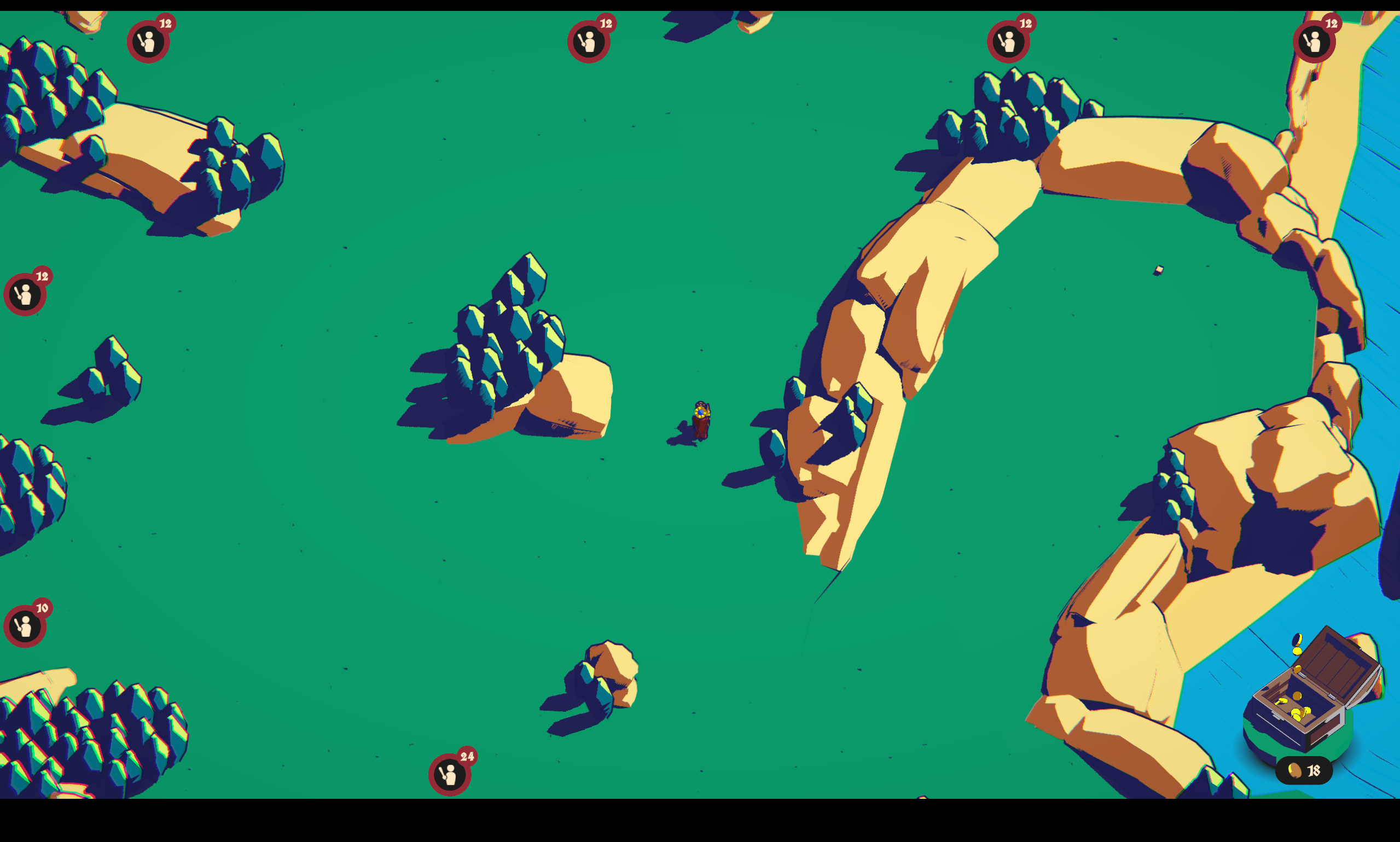
# Monday, November 4

* Update the object pool

Object pool is responsible for spawning enemies down a given path. Need to change to accept some new parameters. Object pool needs to accept a start delay and delay between waves. Also, SerializeField an array of Wave scriptable objects. Foreach wave scriptable object, use a spawning coroutine to yield start delay before executing, iterate through each object in the wave and delay each of them by the waveSO’s enemy spawn delay. Enemies no longer need to be instantiated repeatedly, so they can be destroyed instead of disabled.

Object pool will no longer be independent of spawning enemies. Instead, a game manager will act as a middle man for all the object pools to communicate with each other to coordinate a wave, and the game manager will tell the scripts when to execute a given wave.

The game manager will also display the waves to the player. This will be done very similar to Thronefall. Just copy their homework. This is what they did:



This will help the player get an understanding of what enemies to expect, how many to expect, and where to expect them to come from.

* Create Wave Scriptable object

WaveSO will have an array of game objects that the player will set up in the unity editor. The player will also setup the delay between enemy spawning.

Example Wave:

10 rams that spawn with 3 seconds between each other.

1 ram, 1 flying dude, and 1 quick dude.

* Create a game manager to determine end of level.

Need to get the most number of waves from the scene. Since each object pool will have a number of waves attached to it, it should be easy to get the most number of waves. Then, listen for the user to begin the wave. Once a wave is over, the object pools should wait to play their next wave until the user is ready again.

# Tuesday, November 5

Couple of changes to be made. Need to update the trail pathfinding to allow splitting trails to be taken. This will require a new algorithm to calculate the “path of least resistance”. Heres the logic:

* Each tile node in the trail will be scored 1 point. This means, by default, the enemies will follow the shortest path to the objective.
* If the player were to build something like a wall in the trail, the enemies in the trail will be notified to recalculate the “path of least resistance” and continue accordingly.
* Note: once an enemy chooses one of the forks in the trail, it SHOULD NOT BE ABLE TO PATHFIND BACK TO THAT FORK AND WILL ONLY LOOK AT THE TRAIL AHEAD.

How to calculate the path of least resistance?

1. You start at the beginning coordinates of the trail. Mark this the first path. Iterate through the path, counting the “resistance” of the trail until you reach a scenario in step 2.
2. For each tile node in the trail, check the tile’s neighbors for trail nodes (exclude the previously used trail node). If two or more neighboring nodes are a trail node, then the system will note that this is the end of the branch, store it in the system and return the branch’s “total resistance” after determining the resistances of the neighboring trail nodes.
3. Then, the system will iterate through each of the neighboring trail nodes, repeating the process from steps 1 and 2.

(See “Path of Least Resistance.docx” document for the algorithm process. Just ignore the bottom trail)

# Wednesday, November 6 – Friday, November 8

Time to learn voxel art design. Take the 1-hour course in GameDev.tv about voxel art.

* Using what was learned in the course, make some new path models for the splitting pathways (**T** paths and **+** paths). Try to keep the colors the same as the game, or start from scratch.
* Create new tower, etc. models
  + Cannon
  + Mortar
  + Wall
  + Spike trap
* Create new Enemy models
  + Barrel
  + Goblin
* Create Background Models
  + Rocks/Boulders
  + Lilypads
  + Flowers
  + Seashells
* Add lively models
  + bugs
* Challenge Model: Create ripple effects with the water model
  + A splash effect
  + A tide effect

# Saturday, November 9 – Monday, November 11

Time to implement the voxel art into the game.

* Create tower and enemy prefabs to use in the game.
* Add the background object models into the game
* If I got around to it, add the water ripple effects into the game. This will require some coding. Basic idea: player presses on the water tile and the tile tells all the neighboring water tiles (top left, middle, and right. Middle left and right. Bottom left, middle, and right) to cause the ripple.

After creating the prefabs, it’s time to redo how some code is processed. The first step will be to set up a game canvas for the player to select the tower they want to create. Restrictions will have to be created to prevent the player from making towers where they shouldn’t (i.e. spike trap should only be placed in the trail, ballista cannot be placed on a trail or in the water.). This will require a new scriptable object

Tower Scriptable Object

* Tower scriptable object will store the tower prefab, the types of tiles it is allowed to be placed on, the gold cost of the tower, the types of enemies it can attack (ground, water, air), its health, and other important information regarding the tower.
* Let’s say later down the road I decide to add upgrades to the tower. The tower scriptable object will store this information as well as the logic needed to make the visual changes to the tower prefab necessary. The scriptable object will also want to store important information like the gold cost of each upgrade path and the updated prefab model.

While at it, I should create an enemy scriptable object similar to the tower scriptable object

Enemy Scriptable Object

* EnemySO will store the enemy prefab, the type of enemy it is (ground, air, water type), its health, attack power, upgrade models, and any other additions I make to the game.
* Similar to the towerSO, if I decide to add mutations to the enemies, all that information should be stored in the enemy’s scriptable object.

# Tuesday, November 12 – Thursday, November 14

It is now time to refactor code. Scripts will need to communicate with the scritable objects to get the information they will need. This will take some time to complete and something is bound to break.

After refactoring the code, it is time to finish the UI for the towers that the player can place and allow the player to move the camera around the world. Here’s a basic idea to get started:

* Create a new script called CameraMovement that listens to the player pressing right-click. The game will calculate the mouse’s world position, take the x and z coordinates (don’t want to mess with the zoom of the camera, so no y coordinate), and LERP the camera from its starting position to the end position.
* If the player right-clicks on a new location, the camera should stop its LERP and calculate the new LERP.
* If the player scrolls out, the camera’s y-position should move upward to expand the game view until the max height of the camera is reached, and vise versa.
* Lastly, if the player holds down right-click, the camera should move in unison to the mouse’s change in position for the x and z coordinates. This will also cancel any LERPs the camera may be doing.

# Friday, November 15 – Friday, November 22

Time to create the levels. Here’s a breakdown of what the levels may encompass:

Level 1)

* A basic path from the start to the end of the level. The player can place ballistas and other towers that I feel should be in the very first level
* Enemies will be really simple. Some rams, some barrels, and maybe a special enemy the player will have to concern themselves with.

Level 2)

* Introduce splitting trails. Now, the player will have access to walls they can place on the trail, and they will have to make decisions on how they want to have the enemies pathfind. Waves should abuse the short trails if the player decides to neglect this feature. This can be done by making fast enemy types and restricting the number of towers the player can build on the short path.
* The level should introduce new and stronger enemies that hurt a lot, but move slower. This will balance out the longer path with the enemy’s health.

Level 3)

Enemies will find their own path. The player will have to make strategic decisions to hold choke points and manipulate enemy pathfinding. This level should introduce a new enemy that fires back at the towers while heading towards its objective.

Level 4)

A mixture of trail and own pathfinding will be used in this level. This should incentivize the player to place their towers strategically to cover as much ground as possible since they cannot rely completely on chokes nor can they rely completely on walls

Level 5)

This level will be a rather longer level and will introduce a new structure to the game: gold mines. Gold mines will be susceptable to certain enemy attacks and the player will want to defend their gold mines as much as possible.

This level will require some reprogramming to allow enemies to prioritize building structures. This level will also introduce two new enemy types: one enemy will always target gold mines before targeting the objective. The other will always target the objective.

Level…

Level 10)

To pay homage to the original setup of the object pools, I think every 10th level will have a different setup than the rest. The waves will continue on without the ability to control the waves. This means the player will have to make reactions as the game is being played out without the opportunity to prepare between waves.

This level will end with a “boss” enemy. I am thinking of an enemy that will destroy anything it can reach while heading towards the objective.

# Saturday, November 23 – Saturday, November 30

This will mark the near-end of Realm Rush. Final features like persistent game data should be implemented here. This will also be the time I fix any programming issues I may be having as well as tidy up my game.

I should also spend this time to create main menus, tutorials (if I feel like it), a way to quit, pause, and resume the game, a “logbook” for everything the player encountered, and other QOL stuff in basic games.