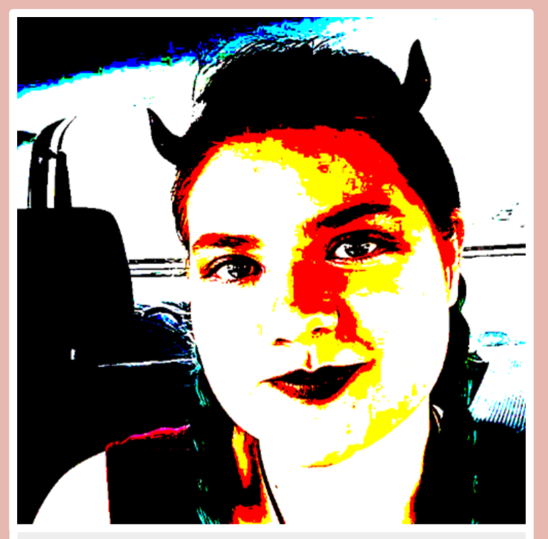

Autumn Hart

Donghyun Kim

Ghias Padmakoesoema

BoOwOo Studio

Dead End Dungeon

OSU, CSE 5912, Dr. Crawfis, Spring 2021

Dawson Pike

Anders Santus

Colby Williams

The Overall Game

- Theme decisions: 1-5: traditional, 6-10: overgrown, 11-15: lava, 16-20: lovecraftian
  Cinemachine for dynamic camera functionality
- Minimap implemented via render texturing and displaying a sprite attached to the Player
- A unique boss encounter is found at floor intervals of 5
- Hazards exist through the dungeon, which damage the player on contact
- When the player enters a portal, the player's vision is blocked as the old floor is deleted, and a new floor is requested
- The pause menu freezes the game, and allows the player to navigate between scenes
- A global Mixer controls the volume of music and sound effects
- The player and props each have Point Lights that light the dungeon, and a fog exists to limit far visibility

Dungeon Generation

- Easy DG (store asset) is used to generate level floorplans
- LevelManager SO for each dungeon theme customizes values and prefabs to give to Easy DG
- LevelSpawner places the Player into each level and generates:
  - # of platforms depending on wall count
  - Props and enemies depending on the tile count
  - Chests to preset value
  - Navmesh for enemies
  - Floor advancement portal
  - Sets dungeon theme depending on floor field count
- LevelSpawner checks the tags of platforms during prop placement to mitigate vertical prop clipping in the event that shorter platforms generate under taller platforms (this can happen because platforms are larger than floor tiles)
- Props have trigger collider checks for collision with other objects to mitigate clipping

Items & Chests

- GroundItem script attaches to game models and holds their respective SO (ItemObject)
- ItemObject contains the item's sprite, whether or not it's stackable, its description, and a method for creating an Item
- Item is instantiated with ItemObject data and also stores the ID, buffs, weapon type (sword, bow, etc.), and amount to receive upon looting (ex. Loot 10 gold)
- Item assigns a random attack type (fire, ice, etc.) to weapons, which has interaction with Player animation system. Weapon types also interact with Player animation system
- Item pickup and selection is done via raycasting
- Raycasts enable/disable item's Outline script (store asset), which adds/removes a shader to a model's mesh
- Chest script randomly picks two items it'll contain and opens/closes on trigger interaction with Player
- Base class attributes seen during character selection are created as ItemObjects and are saved in a JSON file

Inventory System

- InventoryObject inherits SO, has functionality methods for the system, uses a JSON file for saving between game sessions, and has an instance of the Inventory class which then has an InventorySlot array
- InventorySlot class holds one Item, keeps track of what items are lootable per slot, and communicates between the UI and InventoryObject
- "Database" SO is used to assign items IDs
- Item data is contained in SOs that are attached through a script that is then attached to the 3D models
- Delegate methods update the UI and the Player's attributes array when the inventory class methods to delete, swap, or add are called
- A second InventoryObject with no UI is created during the Character Select scene and has one slot for storing the attributes of the class/model the user selects, making attribute passing between scenes even without the Player object easy

The Town

- BuildingGridController is main game manager (currency, permanent attribute bonuses, saving/loading)
- BuildingController manages each building (type, level, sell value, upgrade cost, "adjacency matrix")
- Grid changes update records in PlayerPrefs (building positions, building types, permanent attribute bonuses)
- UI changes communicated to TownMenuController (currency, buttons)
- Bonus permanent attributes determined by number of taverns and buildings adjacent to taverns
  - BuildingGridController recalculates and passes adjacency matrices to affected BuildingControllers
- ShopManager handles shop mechanics (shop inventory generation, purchasing items)
- Explore mode drops a copy of the Player Prefab into the world, hiding UI elements appropriately

Enemies

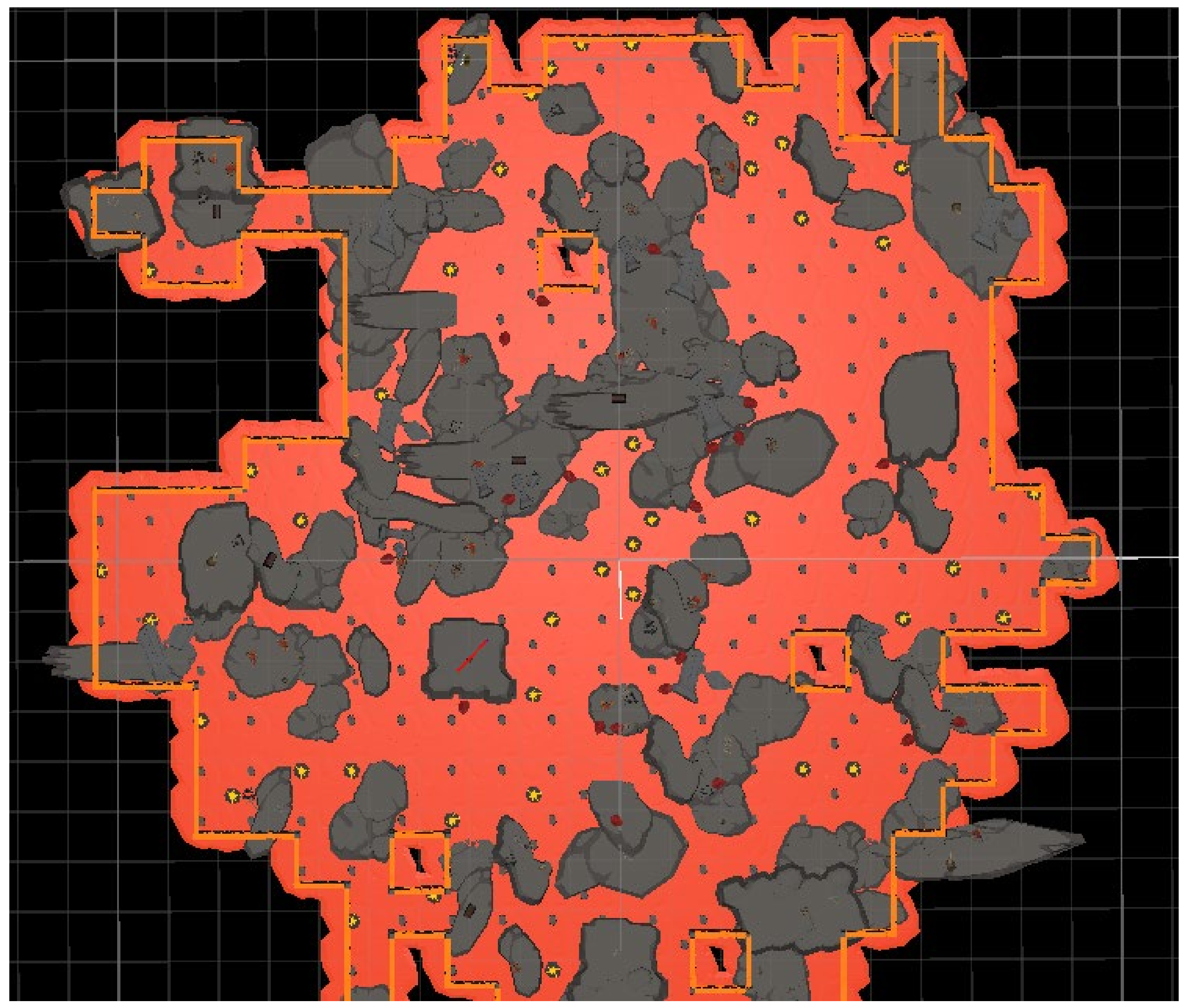
- MeleeEnemy and RangedEnemy inherit AbstractEnemyController and implement custom behavior specific to the general enemy type
- Each enemy has a SO containing specific behavior
- EnemyPool tracks all enemies and determines their status based on distance from player
- AbstractEnemyController checks for collisions with "Weapon" tag, calls Attack method on the weapon to calculate damage, and invokes DealDamageCommand on itself using the calculated damage
- Melee enemies have a disabled collider that is enabled during their attack, ranged enemies instantiate projectile prefabs, and the colliders are tagged with "EnemyAttack" to group them
- Player class checks for collisions with "EnemyAttack" tag, determines the damage from the enemy, and invokes DealDamageToPlayerCommand with damage
- Dungeon uses RandomEnemySpawner to instantiate random enemy prefabs from provided list

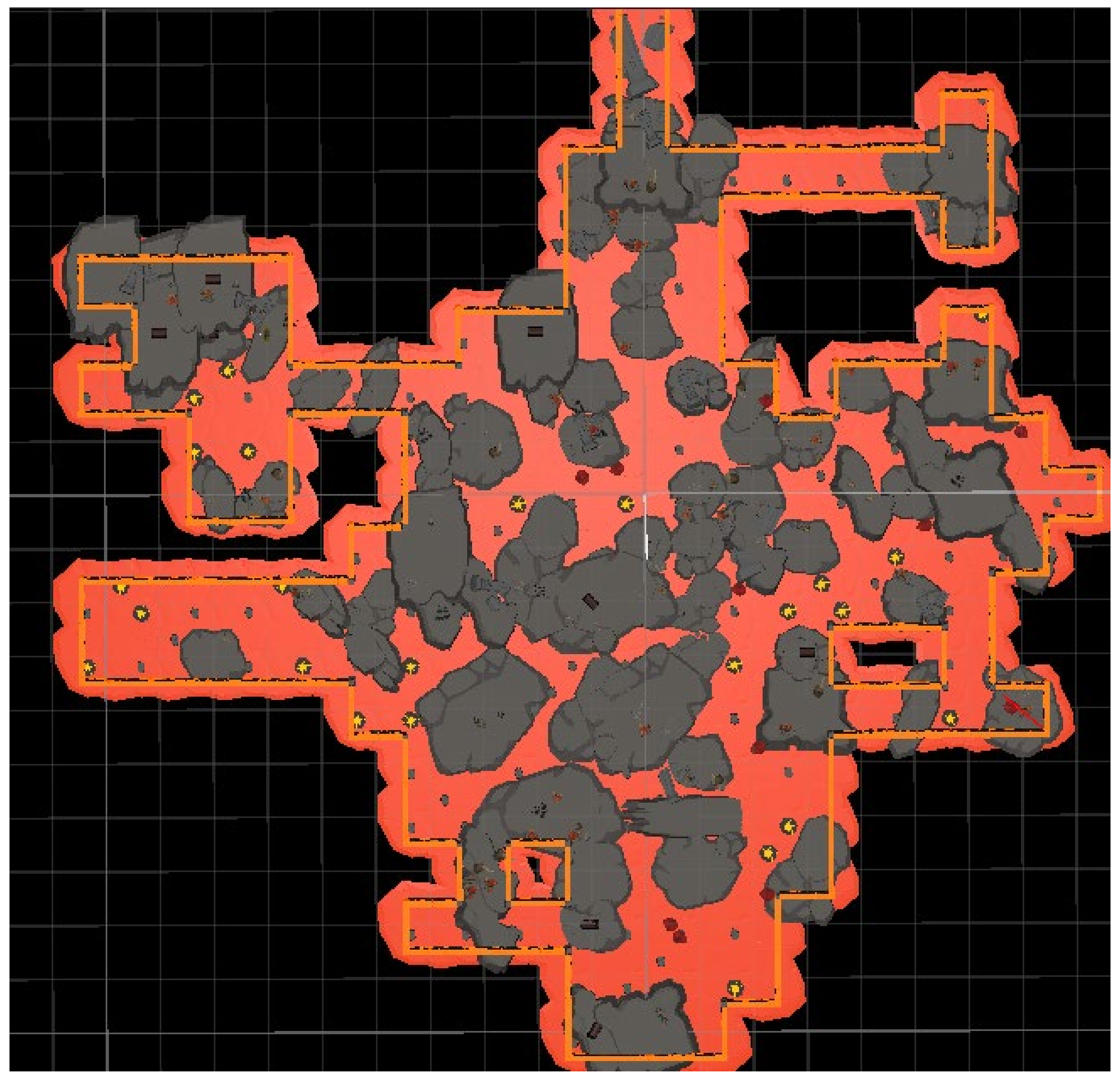
Player Classes

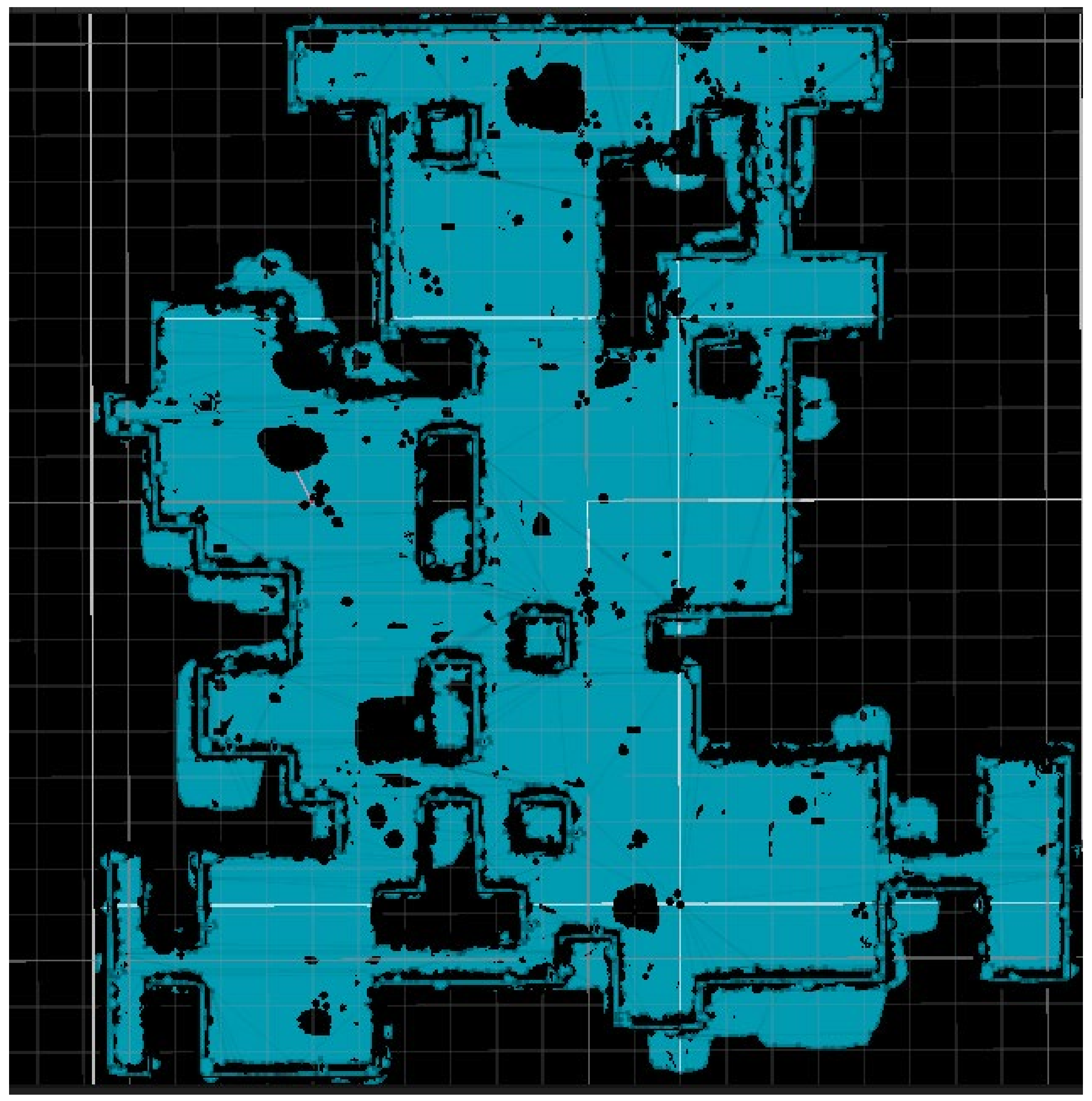
- Class abilities each have their own scripts
  - On pressing their activate key, a BroadcastMessage() is called onto the player for the ability classes with "OnAbilityActivate()"
- Command pattern uses for class abilities
  - CommandInvoker attaches to an Invoker object in the Dungeon scene and instantiates a queue of commands
  - Various scripts add commands (ex. DealDamageCommand) throughout gameplay
  - CommandInvoker executes the logic within each ICommand class and then dequeues them
  - Over half of class-specific commands implemented (5/7)
  - Example: DamageGOsInSphereAreaCommand used for ability area damage (ex. mage's fireball attack)
    - On execute: enqueues DealDamageCommand for each enemy in the overlapping attack area

Animations

*Any animation not mentioned is a store asset
- Player can move 8 different directions
- Player can roll 8 different directions
- Player can jump while moving
- Player has death animation
- Player attack system is handled by player's current weapon and attack type
  - Each weapon has different cool down, and different animations
- Player animations are controlled by player animator
  - Player animator is composed by 2 different layers
  - Multi layer system allows player to do 2 different animations at one time.
    - Player can attack while moving
- Slime animations include walk/idle, attack, getHit, and die


Lava Generation 1


Lava Generation 2


Navmesh View