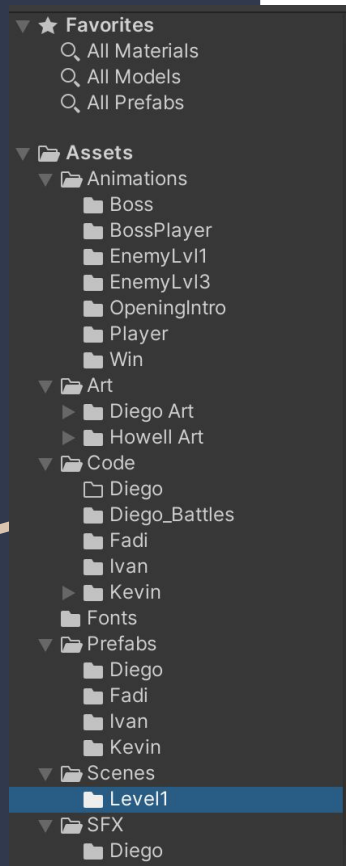
The Legend of Densmore

25-Bit LLC

The Software

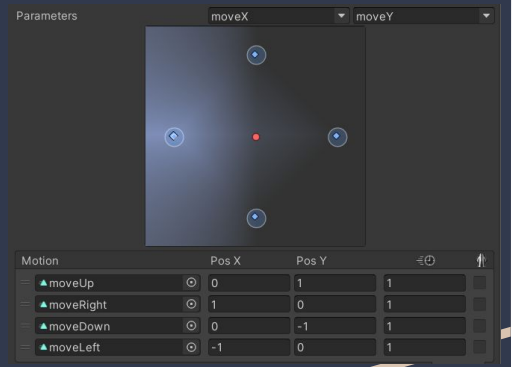
- The Unity game maker software was used to construct the game
- c# was the language used in each of the codes
- While Unity utilizes classes, and they were used, derived classes are not implemented, resulting in the use of many public variables in order to affect player and enemy stats and states
- Unity's coding format works similarly to Arduino with each asset
- Unity allows for assets to be overlaid others smoothly so that items such as audio files can be implemented into the program without the necessity of extensive coding

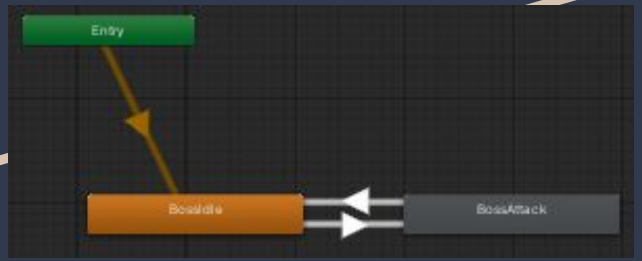
General Design



- The game is a top-down 2D dungeon crawler with turn based combat
- A collection of Prefabs, Codes, Audio Files, and Art/Animations were worked on separately and systematically implemented
- Each member designed a part of a “scene” and combined their part with another member

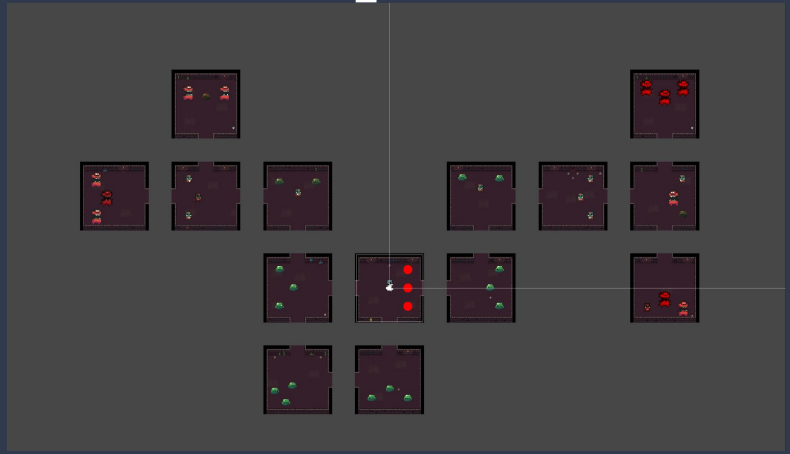
Animations

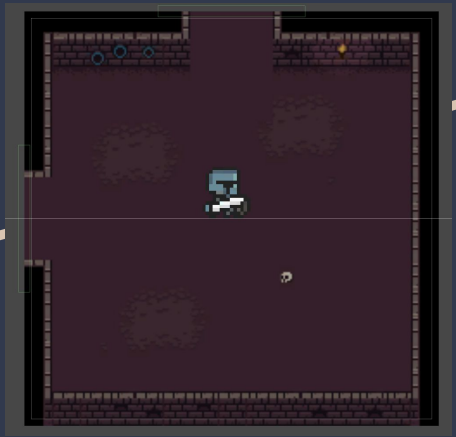




- Animations are done through state blend trees
- It begins at the top of the tree with the animation for idle which requires the user to have 0 inputs in the x-y axis
- For the boss, it will have a timer for attacking, once it attacks, the BossIdle will transfer to the BossAttack state instantly. After the attack, it changes its state back to idle briefly, and continues it's cycle
- The diagram on the right shows how the parameters of movement in the X and Y direction result in its respected animation. Unity will figure out which animation to do based on the actuation points on the graph
- The animation state blend tree is constantly updated every frame when the Update() function is activated

The Map & Rooms



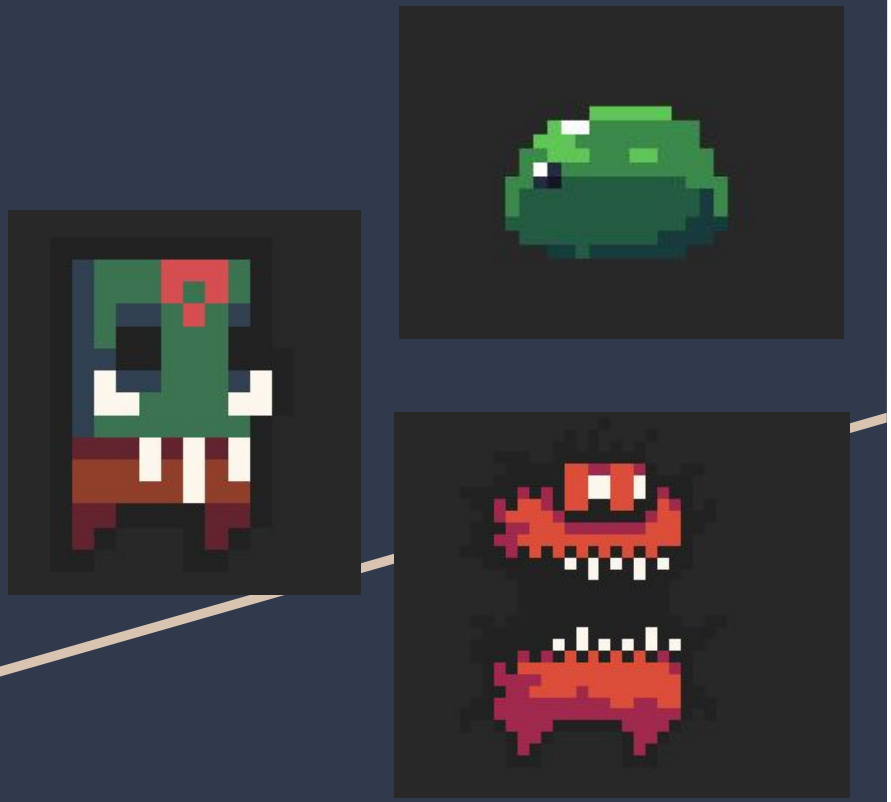


- The map was constructed using a simple square template for each room
- Each variation of room layout (including where the doors are) has unique borders to ensure the player does not go out of bounds
- If a player enters a doorway with the bottom of the body entering the small rectangle, the camera will shift to the next corresponding room

Room and Player Interactions

- When a player enters a room a random number of enemies will spawn that the player must defeat before they can go on to the next room
- The room’s state is default uncomplete, but when a player enters the room the state becomes completed and no enemies will spawn there again
- Every room has a chance to drop an item if no enemies are spawned
  - An item can increase the player’s max health, replenish health, or increase damage

Enemy Design



- A maximum of 3 enemies will appear in every room aside from the first
- Each enemy is a randomly assigned level between 1 and 3 (1 being the easiest to defeat, 3 being the hardest)
- A random sprite is also assigned to each enemy
- When an enemy is activated, it’s state is alive and remains that way until the player defeats it
- For every turn of combat, the enemy can either attack or block

Player and Enemy Interactions

- The player will attack all enemies at the same time, but the enemies will individually attack the player
- The player has a chance to dodge all enemy damage and to perform a critical attack which deals double damage
- The enemies cannot crit or dodge
- The player will not move until all enemies are defeated

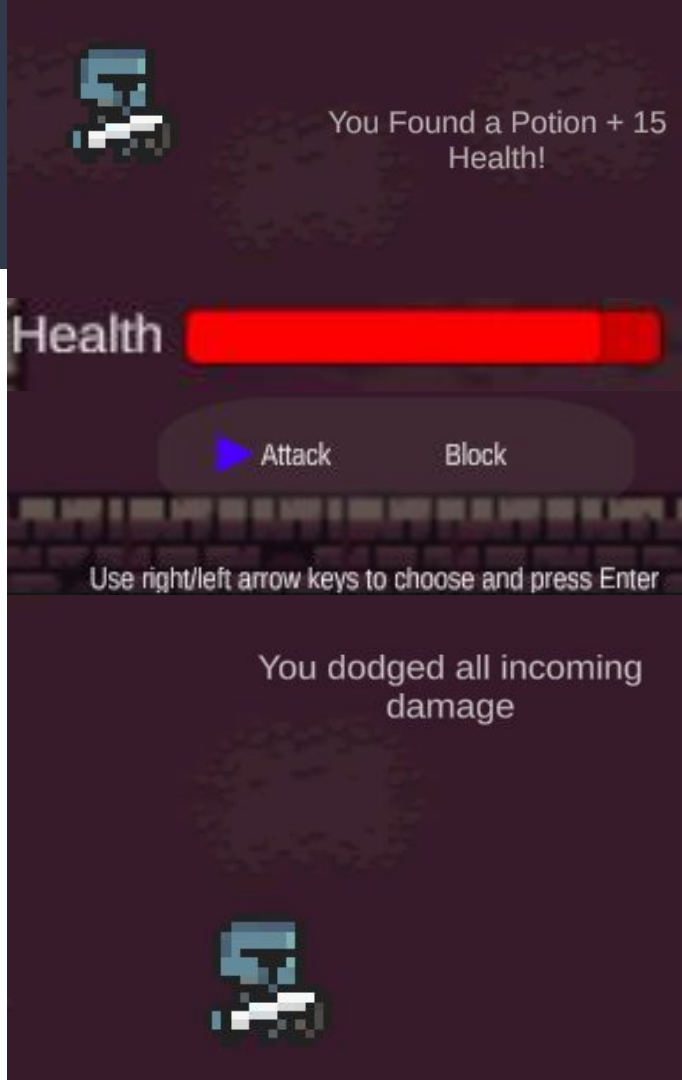
Battling Function

- The battling function begins by simply seeing how many enemies are activated
- Based on the number of active enemies, a different case statement is called to perform battles
- At the start of each case a variable is instantiated to represent the boolean value of if the player will dodge and/or crit, and if the enemy(ies) block
- The player will attack first with either a crit or normal damage and then the enemies will attack after
- If all enemies block the player receives no damage

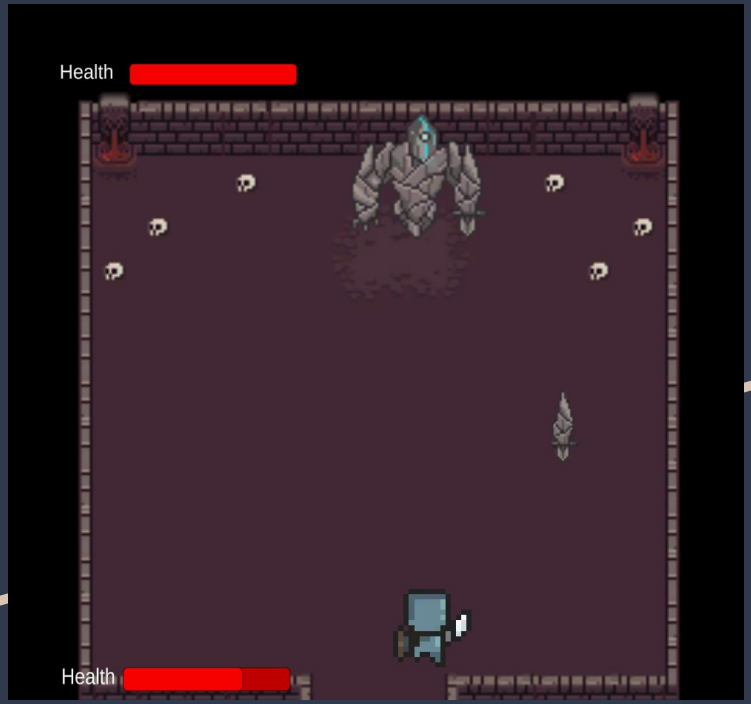
```
{
    if(Player1.GetComponent<PlayerStatus>().player_state == PlayerStatus.PlayerState.PLAYER_ALIVE && (Enemy1.GetComponent<Enemy>().enemy_state == Enemy.EnemyState.ALIVE))
    {
        enemy1_block = IsBlocking(); //stores the state of if the enemy is blocking or attacking
        player_evade = DoesDodge();

        if(choice.GetComponent<TextChoice>().choice() == 1) //battles
        {
            print("Player is attacking");
            if(DoesCrit()) //if the player lands a crit
            {
                popup.GetComponent<PopupManager>().create("You landed a crit!");
                print("Player landed a crit");
                if(enemy1_block) //if the enemy is blocking while the player crits
                {
                    Enemy1.GetComponent<Enemy>().enemy_health -= damage; //half damage * 2x mulitply cancels out
                }
                else
                {
                    Enemy1.GetComponent<Enemy>().enemy_health -= (damage * 2); //if not blocking, the enemy receives 2x damage
                }
            }
            else //non crit attack
            {
                print("no crit");
                 if(enemy1_block) //if the enemy is blocking
                {
                  //  popup.GetComponent<PopupManager>().create("You dealt " + damage.ToString() + " damage");
                    Enemy1.GetComponent<Enemy>().enemy_health -= (damage / 2); //half damage because the enemy is blocking
                }
                else
                {
                  //  popup.GetComponent<PopupManager>().create("You dealt " + damage.ToString() + " damage");
                    Enemy1.GetComponent<Enemy>().enemy_health -= damage; //if not blocking, the enemy receives normal damage
                }
            }
        //enemy turn to hurt the player
        if(player_evade)
        {
            popup.GetComponent<PopupManager>().create("You dodged all incoming damage");
            print("Player dodged");
        }
        if(enemy1_block)
        {
            print("enemy is blocking");
        }
        else
        {
           // popup.GetComponent<PopupManager>().create("The enemy is attacking!");
            print("enemy hurt player");
            Player1.GetComponent<PlayerStatus>().player_health -= Enemy1.GetComponent<Enemy>().enemy_damage; //player takes set damage from enemy if the enemy isn't blocking
        }
        player_block = false;
        choice.GetComponent<TextChoice>().reset();

        }
        else if(choice.GetComponent<TextChoice>().choice() == 2) //defend
        {
            if(enemy1_block)
            {
               print("both are blocking");
            }
            else
            {
               // popup.GetComponent<PopupManager>().create("The enemy is attacking!");
                print("enemy is attacking player");
                Player1.GetComponent<PlayerStatus>().player_health -= (Enemy1.GetComponent<Enemy>().enemy_damage / 2); //if player defends, only takes half damage
            }

            player_block = true;
           choice.GetComponent<TextChoice>().reset();
        }

        else
        {
            return;
        }
    if(Enemy1.GetComponent<Enemy>().enemy_health <=0)
    {
        choice.GetComponent<TextChoice>().deactivate();
        Player1.GetComponent<PlayerMovement>().ismoving = true;
    }
```

Player Choice & Popup UI

- The choice class activates at the battling state and listens to user input at every frame to select a choice
- The popup manager class has a public function to instantiate popups that fade within a few seconds for special events
- The health bar scales every frame according to the player or enemy health



The Boss



- The boss room is a completely different battle system with a simple shooting mechanic and a change of music
- When the player reaches a specific point on the map they are sent to the boss room which is detached from the rest of the map
- The player’s health is restored as a result of the change in combat for fairness
- The boss’ level and health are completely separate from the common enemies
- If the player defeats the boss, a victory screen is presented, if they lose, a loss screen is shown