MINI-MANUAL

Every time you play our text-based adventure game, you are plunged into a brand new procedurally generated level layout. Progress through the rooms using the “move” command. The player can travel in the four cardinal directions (n, e, s, w). As you travel through rooms, use the “inspect” command to check your stats, inventory items, floor loot, and (living) enemy loot. You can use the “equip” command on a weapon or armor in your inventory to equip it on your character. Armor provides invaluable resistance from the monsters! If ever you get lost, use the “location” command to learn your exact coordinates. You’ll encounter a variety of enemies throughout your journey. Be brave! You won’t be able to leave a room while there’s still an enemy alive. MUST SLAY THEM ALL! To enter combat mode, use the “attack” command. From there, you can choose to “attack” or “consume”. The “consume” command allows you to eat consumables from your inventory to gain health points. Yummy AND healthy! Once you slay an enemy, they can drop valuable loot on the floor! If something catches your eye, use the “take” command to pick it up. Caution! If you overfill your inventory, you won’t be able to move anymore. Use the “drop” command to empty items from your backpack. To progress to the next level, you must slay the boss of the current floor. The boss is located in a random room so you must hunt him out! Once you slay him, you may advance to the next floor with the “continue” command. If ever you need help in game, just use the “help” command. This will give you a quick reminder of the game’s commands. The goal is to progress through the dungeon as much as possible. How deep can you get? Have fun adventurer!

DESIGN-GUIDE

**Levels and Tiles**

To create a randomly generated dungeon, we decided it best to create a class representing a level, and then generate it every level to make it easier for us. It turns out, it wasn’t as easy as it seemed. A level is a graph, where vertices can only point in the four cardinal directions (maximum outdegree and indegree of 4), and where the center point is an arbitrarily assigned center tile. A tile is a single ‘room’ in the level containing information about the four outgoing directions, the tile and direction that links to it (there may be more than 1 link, but this is the first one for a tile), any enemies present, any loot present, the tile type (Start, Enemy, Boss), the location relative to the center tile (a combination of some or multiple cardinal directions) and the path to get to the tile, which is the path that the level generator took to create it. To find the location, we needed to create a ‘location parser’ that would turn a path (e.g., nse) into a relative location (e.g., e). The \_\_str\_\_() representation for the tile prints the location (if it isn’t the center tile), the tile type, the enemies, the paths available, and the path to get to the tile (so that the player can find their way back to Start if they’re lost). The level creation was much more complex to code. Since we wanted a randomly generated dungeon WITH tiles that could have more than one indegree, we needed to use recursion and have a way to prevent tiles from being created on top of one another. Simply, the level generator is given a tile, then has a 50/50 chance of creating a tile in a cardinal direction. If it does create a tile in that direction, it then runs again for that tile, and so on, until it reaches the arbitrarily set max depth, which I found to be good around 3. To prevent tiles from being created on top of one another, a dictionary must be created in \_\_init\_\_(), keeping track of every location as a key and the path to that location as a value. If a tile would be created in a given location, it must first check if that tile is occupied, and, if so, it must instead create a link between the creation tile and the existing tile. To create that link, we must find the existing tile, which is where the aforementioned dictionary is used. A simple algorithm is then used to locate the existing tile using its path. Finally, the level generation is finished with a boss tile creator, which does a random walk until it reaches an empty location and creates a boss tile there. The reason for this random walk is so the boss room doesn’t prevent a player from going down a given path.

**Name Generation**

We made a list of adjectives, armor names (based on location (head, torso, arms, legs)), weapon names, consumable names and enemy names. We then implemented a way to create the name of an enemy, consumable, weapon or armor by randomly choosing 0-2 adjectives and randomly choosing the respective category’s name. For example, a weapon could be named Blazing Crowbar, and an enemy could be named Blue Goblin. Adjectives and names have no actual effect on the thing they’re naming, instead they were used as a way to avoid duplicate names. Duplicate names are both boring (you don’t want to just attack goblins on every tile), and they can lead to some misinterpretation of what action the player wants to take (attack which goblin). Consumables do not have adjectives.

**Inventory and backpack**

The inventory has two main storage uses. One, keep track of the weapons and armor that the player has equipped, and two, the items the player has in their backpack. The backpack was initially meant to be an item that a player could find as loot, and was therefore created as a class, but a list within the inventory with some methods would work just as well. The methods of the inventory are to equip items, drop items, check how many open slots there are in the backpack, check how much total damage your weapons deal, check how much total resistance your armor has, and show the items in the backpack. If an item would be equipped where an item is already equipped, the equipped item is placed in the backpack and replaced. These are all used elsewhere, although the only two used explicitly in the game.py code is the equipping method and the dropping method. The backpack has only 3 methods: add item, remove item and check if it is over capacity.

**Items**

Each item type (excluding the backpack) has one main stat which is randomly generated, and the max of the random number generator increases with every stage completed. Armor’s main stat is resistance. It negates enemy damage equal to the resistance. Weapons’ main stat is damage. Finally, consumables’ main stat is healing, and it heals the player. Backpacks have a maximum size.

**The Player**

The Player of the game is created from a class. The class initializes the player’s max health, base damage, inventory, number of enemies killed, experience points and level. When the player reaches enough experience points, they level up, with their max health and base damage increasing by 1, and their current health is set to the maximum. The methods for the player are for increasing xp and levelling up, calculating damage dealt and damage taken (which does all the math with armor resistance and deals damage to the player), checking if the player is dead, consuming an item, and finally showing loot, current health, max health, damage, resistance, and name. The str method shows all of these except loot and including open backpack slots.

**Enemies**

Enemies are very similar to players, except instead of having an inventory they have a list of loot that they transfer to the tile they’re on when they die. The amount of loot is randomly generated but the maximum amount of loot increases when it’s a boss. There are two types of enemies: Normal enemies and bosses.

**The Game**

The game generates a Player and a level and makes the current tile the center tile. The current tile denotes the tile where the player is. A while loop then runs the game. The while loop first checks if the player is dead, and, if so, offers to start a new game or quit. Then, the player must choose from a list of commands, then an if statement acts based on the command. These if statements do checks on the command to see if it’s valid, then act upon it using methods. For example, if the player writes ‘Drop Banana’, it will find its way into the drop if statement, then if banana is an item in the backpack, it will execute the inventory drop method. The player starts on a tile with no enemies and must then go through a series of tiles to reach the boss, as described above. The only if statement that does not use a separate method is the combat command, as it’s not a part of any other system or class. Within combat, a player’s actions are limited to a few combat-related actions, and the whole combat loop is separate from the main command loop. If either the player or the enemy dies, the player is notified that they killed the enemy, any loot from the enemy is dropped, and the player goes on their merry way. If the level boss is killed, the player is then notified they can go to the next floor. If they choose to do so, a new floor plan and current tile is generated, and the player starts the again, but this time with a higher enemy difficulty. The commands (and sub commands) the player can do during free movement are: move (n,s,w,e), equip (armor or weapon from inventory), attack (enemy), consume (consumable), help, continue, inspect (floor, enemy, self or inventory), take (item from floor), drop (item from inventory), location, and quit. The only commands of further note are location, help and inspect. Location will print the tile the player is currently on and the paths they can take. Help will print a help message and a list of these commands. Inspect floor will allow you to see the loot of the tile, although it isn’t necessary for the take command. Inspect enemy will check the loot of the enemy. Inspect self will check the player’s own stats. Inspect inventory will print all equipped items and all items in the player’s bag.