Liu, Andrew
aliu3
15-112 Term Project
11/20/19

Project Description:
My project will be a puzzle/adventure video game based on the 2008 2d platformer “Spelunky.” It features a character stuck in a cave, who must creatively use tools to maneuver past obstacles and find a way out under limited lighting. It will also feature random level generation, in which multiple random rooms are connected together to guarantee a winning a path.

Competitive Analysis:
Spelunky was an open-source indie game originally made for windows PC, and released as *freeware* by its creator Derek Yu. It was an early influence for many later “roguelite” or “roguelike” games, which are essentially tile-based dungeon-crawlers centering on exploration and survival. However, Spelunky was built to be a lot more simplified, neglecting the turn-based gameplay and heavy NPC interaction. I’m aiming to preserve this simplicity by not having all of the many features of the original game, but *changing it up by restricting the lighting, so the game is mostly played in the dark.* The player will have a limited number of torches which run out after some time, and will have to find additional lighting in chests around the stage in order to progress further forward. This way the gameplay will be more true to the idea of “spelunking” a cave, which was the idea I built this project around, rather than Spelunky itself (which I’ve never played myself).

Spelunky’s open-source status means there are a lot of similar remakes out there that I can study--either by independent developers and amateurs such as myself, or larger ones such as the 2012-13 Xbox and Playstation versions. However, its game physics and animation are pretty straightforward and shared with most 2d platformers, so I will be developing my own engine for that using PyGame. What I will be investigating from Spelunky renditions is its key feature of *procedurally generated* stages, meaning that each playthrough of the game is unique. My method of generating random stages with a guaranteed exit path will definitely draw from these renditions, but I will modify and implement it accordingly using python and Pygame.

Structural Plan:
I am still looking into how to use and run multiple Python files in a directory in tandem (using import), but for now my game may be split into distinct chunks of code:

1) **Block superclass and subclasses:** the environment. Most subclasses of the “Block” object (dirt by default) will be a part of the stage the player can collide with, and must navigate around unless somehow removed (terrain). It may also be something like water or rope, which the player can navigate through, or obstacles that kill the player such as lava or spikes. Finally, there are chests, which contain helpful items.
2) **Stage / generateStage() / generateRoom() / generateChunk():** the following are composed of individual block classes, in increasing order: chunks, rooms, stage (see

algorithmic plan). The methods will procedurally generate these larger objects, each of which encapsulate its smaller components.

3) **Player:** the player contains all collision and other movement/physics data, as well as other necessary parameters (jump height, run speed, etc) and an inventory for holding items. It also contains method(s) to move the rest of the stage (the game is a sidescroller, so the player itself actually never moves) (sprite subclass).
4) **Item:** objects stored in chests or the player which are not tangible / drew on the stage. Unless I have time this will only refer to torches or rope, which provide the player extended periods of light or can be placed on the stage to climb walls, respectively.
5) **Entity (monsters/enemies, or perhaps friendly NPCs:** this will be implemented if I have time, providing an extra level of complexity to the game. Monsters will have a random AI movement, damaging the player on impact, while NPCs will give helpful items or advice to help the player survive (sprite subclass).
6) **Main:** to run the actual game loop and initialize the player, stage, and entities, and check for gameWin or Lose (based on player health or time).

Algorithmic Plan:

The most complex part of this project will probably be in the level generation. I should be able to do this by storing the chunks, rooms, and stage as randomly generated nested lists of numbers, where each number represents a type of block (0s are spaces). Similar to Minecraft, the entirety of the level will be stored in these 1x1 “blocks” (see description in structural plan). Larger chunks will have their own constraints, such as containing a certain arrangement of walls (so the player is not trapped), or a certain number of obstacles, item chests / torches, or friendly and unfriendly entities. These chunks will make up *rooms*, which each have an exit in at least one direction: up, down, left, or right. The player will be spawned on a room on the left edge of the map, and at least one room on the right border of the map will contain an exit to the right, representing the end of the stage. The algorithm will string rooms together in a way that guarantees a solution path, given these conditions, possibly using a form of backtracking. Non-solution paths will then be filled in with dead-end rooms.

Timeline Plan:

TP1: Player movement, physics engine / collisions, animation completed
TP2: MVP with random level generation and lighting system completed, as well as rope and torch items + rope and chest blocks.
TP3: Extra features--obstacles, NPCs and enemies, other game screens, game modes, music and sound effects, and extra interactive items and blocks

Version Control Plan:

I will be backing up my code in the cloud, on my student account’s Google Drive:

Module List:

- **pygame** for game creation, visuals, and audio

- **random** for random stage generation and entity movement
- **math** for calculations