Ruins of Pythonia

Derek Xu, Kaif Jeelani, Anshul Kashyap, Akash Darbha
AP CSA per.2
11/5/20-12/3/20
Screencast
Presentation
Github Repo

Table of Contents:

Overview:

The purpose of our program was to create an educational game that would help intermediate students practice and refine their programming skills in Python. To keep the students engaged while they were utilizing their skills, we designed a game with the theme of an archeologist looking for knowledge about the lost civilization of the imaginary world of Pythonia. The user would play the game by controlling the archeologist and avoiding obstacles. If the user runs into an obstacle, they will get a chance to revive their character by answering a question about a Python topic. If the user gets the answer correct, then their avatar will be revived and they will continue playing the game. If the user gets the answer wrong, then the game will end. At first, we were planning on implementing a series of rooms that would get harder and harder as the user progressed further and further into the game. But as we started developing the game, we realized that the code would be far too complex to handle in the available time frame. So, we decided that there would only be one room for now and that the goal is to get to the end as fast as possible before the walls closed in on you.

Design:

Rough Draft: We originally wanted to have multiple rooms with different tasks, but because of a lack of time, we had to settle for our prototype which is covered in our final draft

Figure 1: The original game created by Anshul 2 years ago

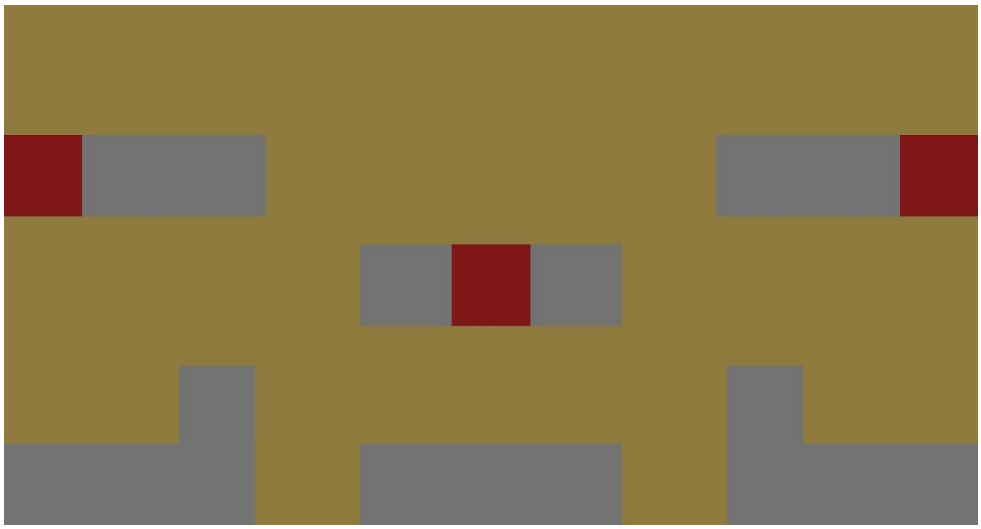


Figure 2: A quick schematic of the monster room should we implement it. Grey means standable tiles, red means deadly tiles, monsters would randomly spawn from the deadly tiles. If you were to kill 20 monsters, you could progress to the next room

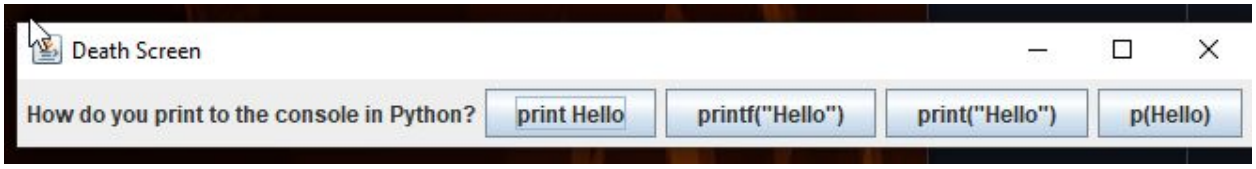


Figure 3: This is an example of a screen that pops up when the user's avatar dies. The user has the opportunity to revive their avatar by answering a question correctly about Python.

Initial Room Design:

Room #	Room Id	Description
0	Title	The title screen with instructions and play button
1	Intro	Very simple room with no puzzles and one monster. You will learn how to move, jump on platforms, shoot a fireball at a weak monster, and read a sign. Sign 1: This sign will contain information on how to print in Python
2	Puzzle1	This room will contain the first puzzle room. There will be a sign in the background. On the bottom of the screen there will be five blocks with "print", "(", "Hello", " "Hello" ", and ")" You will be asked via a sign to step on the blocks in the correct order to print out Hello properly
3	MonsterFight1	This room will randomly contain monsters that start coming for you. If they hit you you lose a heart. You have three hearts till you die. There are 15 monsters in total until you can move on
4	SignRoom1	Here the user will read the runic readings(conditionals) and keep them in mind for the next puzzle
5	DANGER	Here the user will encounter the walls closing in on them, forcing the user to run to the exit as fast as possible
6	BossFight	Here the player will fight an ogre shaman. The ogre shaman shoots projectiles rapidly, and it takes 20 fireball shots to kill it. You then win the trial version
If you die by falling in holes or monsters, you can revive at a checkpoint as long as you solve a Python Question		

Final Draft: As we were designing this game from a small prototype created from Anshul a few years back, we decided that we would only incorporate one room, which happened to be the DANGER room. We originally planned to make logic for infinite terrain, but then we decided against it again in favor of time. We focused more on the revive aspect of the game which was intended to be the most educational part of the game from the start

Create and Test:

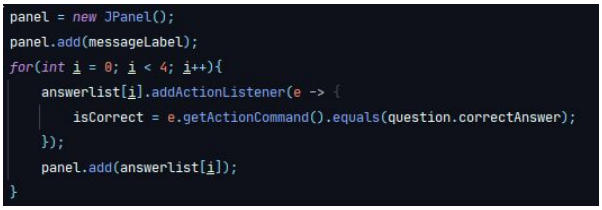
We had a couple of things we had to implement before we began testing. The main things that we had to implement were the map logic, the player controls, collision detection, and player death logic. To implement the map logic, we came up with a unique way of modifying an array of integers and mapping them to the actual map the player interacted with. We would basically go through each individual element in the array and if it was a 0 we wouldn't add a block if it was a 1 we would add a grass block, and if it was a 2 we would add a dirt block. After this we added the player to the screen and made it so that we had key listeners for the arrow keys for player movement. Additionally, we added 4 rectangles to the player which would act as collision detection rectangles and would allow the player to interact with the blocks as if it were in the real world. The final thing we had to implement would be to add the player death logic. We wanted to make the educational aspect of the game to teach the player if they got the wrong answer. As a result, we came up with the idea that if the player dies, a new window would pop up with a question and possible answer choices. If the player pressed the correct answer they would respawn but if they died they would be redirected to a youtube video about the topic they missed. Once we implemented the different features, we tested them thoroughly and had numerous use cases to make sure the game worked as expected in every single situation. For map logic, we played around with different positional values and found a couple of bugs where if we placed a 1 at the edge of a row it would overflow off the screen. We were able to fix that and did not encounter any serious problems other than collision detection between the player and the blocks. Sometimes the player would phase through the blocks and not come up so we had to fix it so that the player would never be able to phase through a block and always bounce up.

Code:

Screenshot of Code	Explanation

//Initializing setting
int[] field = {
2,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,2,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,2,
2,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,2,0,0,0,0,0,0,1,1,0,0,0,0,0,1,0,0,0,
2,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,2,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,0,0,0,0,0,0,0,1,1,0,0,2,
2,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,1,1,2,0,0,0,2,2,0,0,0,0,0,1,1,2,2,0,0,2,
2,0,0,0,0,0,0,0,0,0,0,0,0,1,1,1,0,0,0,1,0,0,0,0,0,1,1,2,2,2,0,0,2,2,2,2,2,0,0,0,0,0,0,0,0,0,2,
2,0,0,0,0,0,1,1,1,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,1,2,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,2,
2,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,1,0,0,0,0,1,1,1,2,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,2,
2,2,2,2,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,2,2,2,0,0,0,2,2,2,2,2,2,2,0,0,0,0,2,2,0,2,0,2,0,2,0,0,2
};
	This segment of the code was how we designed the background of the game and the placement of the tiles. We were able to make a scalable game by implementing a number to block mapping system where 0 represents no block, 1 represents a grass block and 2 represents a dirt block. This way, we would be able to design levels very quickly just by changing a couple of numbers. Additionally, we added a feature where we would change a number and it would automatically scale up the map. After this, all we would have to do is add to our array of numbers to design a bigger or smaller level depending on what we wanted.

public void jump() {
    if(!jumped) {
        if(!doubleJump) {
            velY = -6;
            doubleJump = true;
            currMove = 0;
        }
        else {
            velY = -6;
            jumped = true;
            currMove = 0;
        }
    }
}
	The jumping/double-jumping logic implements if statements to decide if the cat should jump, and what type of jump the cat should make. The if statement which checks if the boolean jumped is not true ensures the code in this method is executed only if the cat can either double jump or jump off a surface. In the if-else statement inside of the outer if statement, it checks if doubleJump is false or not so that the program can decide if it needs to make the cat double jump or jump off a surface. If doubleJump is false, then the cat will make a normal jump by setting velY to -6 and currMove to 0. Furthermore, doubleJump is set to true to signal that if another jump is made before the cat lands on another surface, that jump will be a double jump. If doubleJump is true when the function executes, the cat will make a double

	jump by setting velY to -6 and currMove to 0. Furthermore, the jump variable is set to true so that the cat can not make any more jumps until it lands on a surface.
 	This segment of the code is where we implemented our questions to the user. So as the user committed a mistake and their avatar would die, the user is then given an opportunity to revive their avatar by correctly answering a question about Python. We programmed the game so that the user would be prompted with a new panel of a question with four answer choices available. If the user selected the right answer choice, then they would continue to play the game from where they left off. If they answered incorrectly, then the game would end.
	To help students augment their python knowledge, we decided to implement educational videos into the program. If the user gets a question wrong, the program will open a new tab with a video about the topic along with other related topics(by YCW). That way students can learn new topics while practicing the topics they have already learned about.

Future Improvements:

Due to our constraints such as the time we were unable to develop our project to the extent that we wanted to. In the future, we would like to implement other features in the game. The first feature that we would like to implement is the different rooms available for users. As the user would go further into the game, then they would be able to advance to different rooms, which would be different levels. Within the different rooms, the user would face different monsters and would need to defeat the monsters to advance. We would also like to expand the game by adding more questions that the user could face. Right now, we only have a few questions available for users, and with an expanded set of questions, they would be able to practice a wider range of their python skills. We would also like to make the game less finite so that the game would last longer and be more interesting for users.