Final Report

Project Name: Interactive text-based game on Raspberry PI

Team Members: Ignacio Gonzalez, Timothy Wood, and Andrew Garcia

Introduction:

For our project we decided to design and implement a text-based dynamic game based on Python, which could be run on a Raspberry Pi.

Initially, we were going to make a purely text-based game, without any type of complex mechanics or user movement. But we discovered a library on Python, called Pygame, which allows the creation of immersive dynamic games.

We decided to take the route of using Pygame, as we could apply class concepts like multithreading and multiprocessing in a better way than our original idea for the game implementation.

Our game objective is to explore a dungeon filled with monsters, fight the dragon at the end and save the princess. We also added several different mechanics like healing, chest opening, room transitions, power ups, and timed events with user decisions.

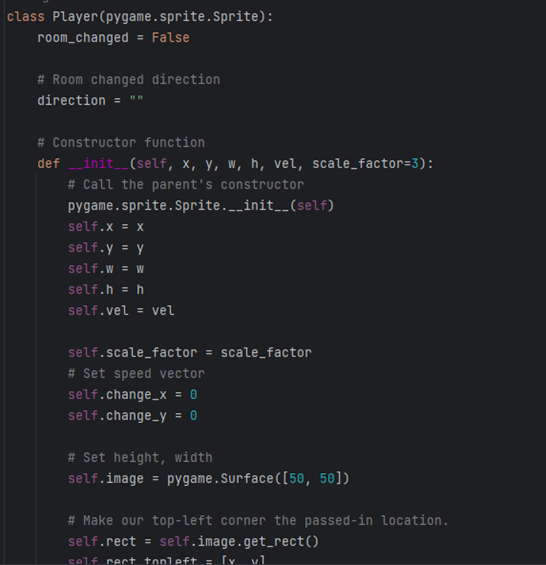
We also have a separate process running at the same time as our main process, in which the map of the game is displayed. So basically, we are running 2 processes at the same time, one for the main game and the other one for the map.

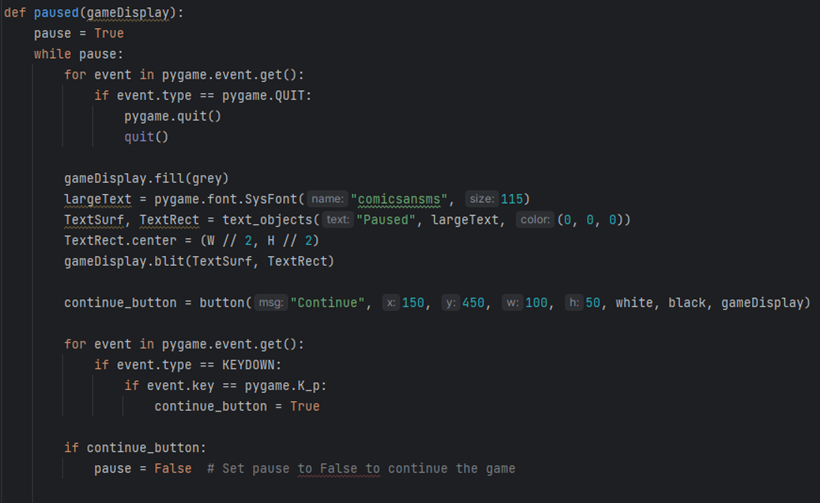
Implementation:

The first step of our program is to create all global variables and all other methods that will operate outside the game loop.

Which includes creating the walls in the room, loading the images, method for pausing game, drawing health bars, etc.

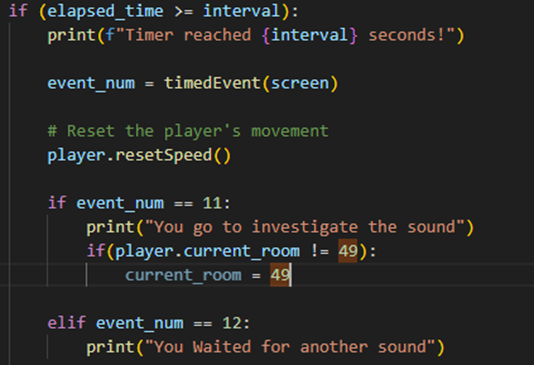
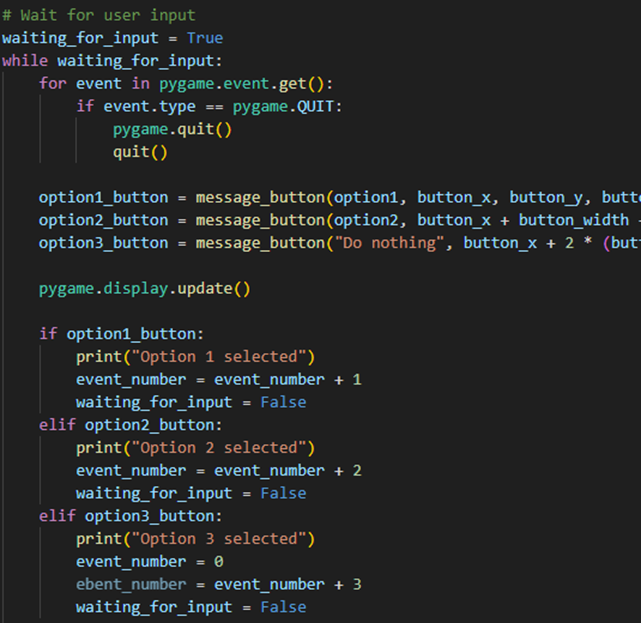
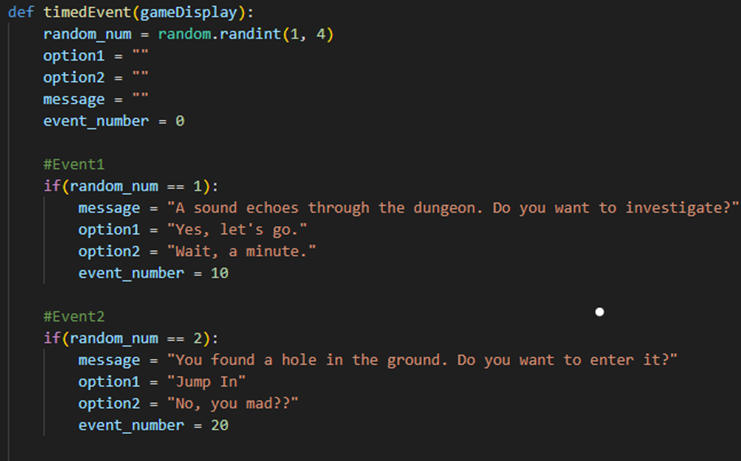
Along with player and wall classes





Implementation:

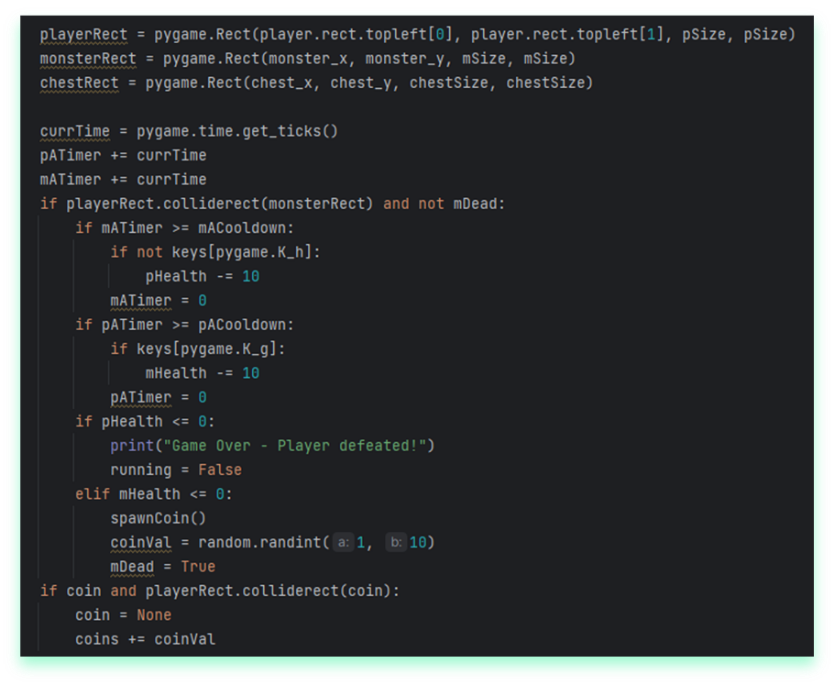
In these sections we handle the events generation and user decisions.

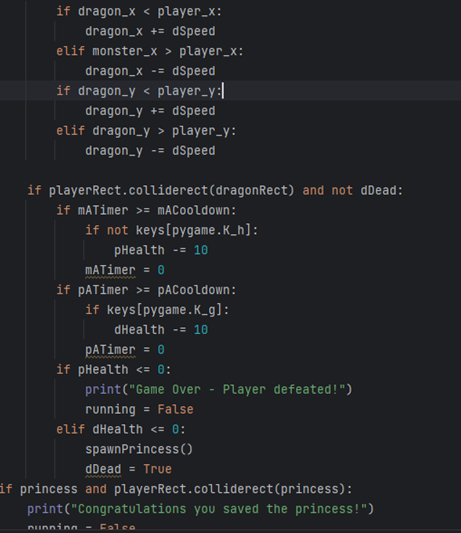
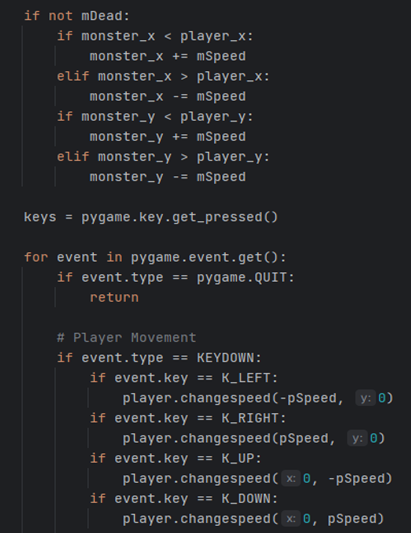


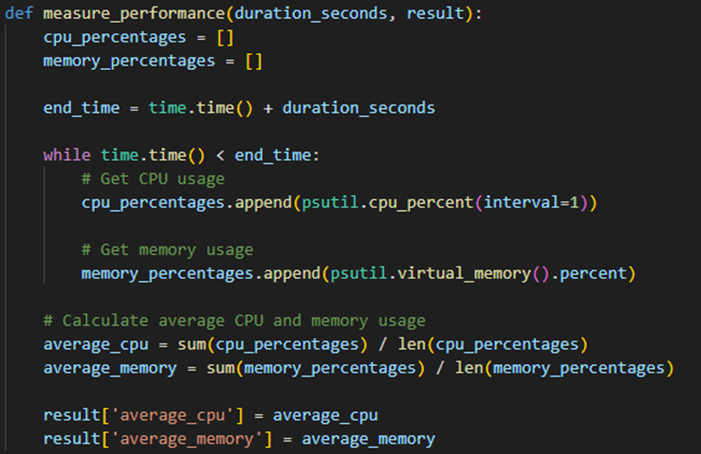
Implementation:

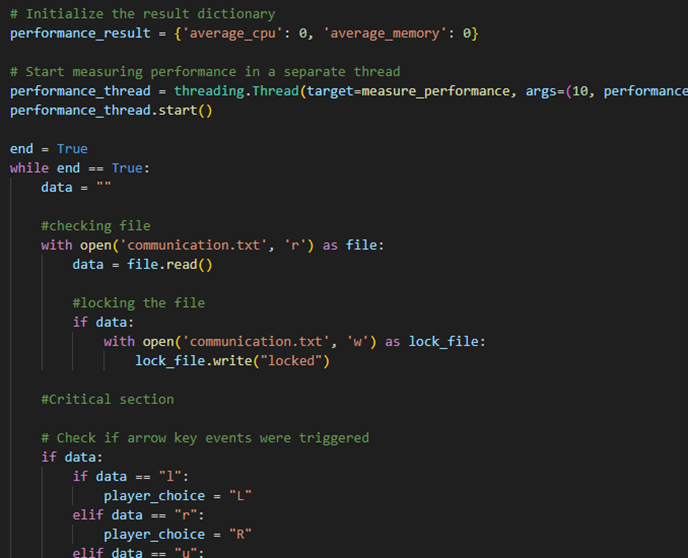
Inside the game loop we have all the movement and collision for the player, monster, boss, coin, chest, and princess. Along with attacking and blocking.

This is where we also draw all our items.





And finally, this is the process handling the display of the map and measuring system performance:

A screen shot of a computer program

Description automatically generated

Challenges

Synchronization and Multiprocessing:

* This was a big challenge as it took a big amount of time to set up.
* Because of the limitation of the Raspberry Pi, we had to remove some features and threads.

Rooms design and Monster Spawns:

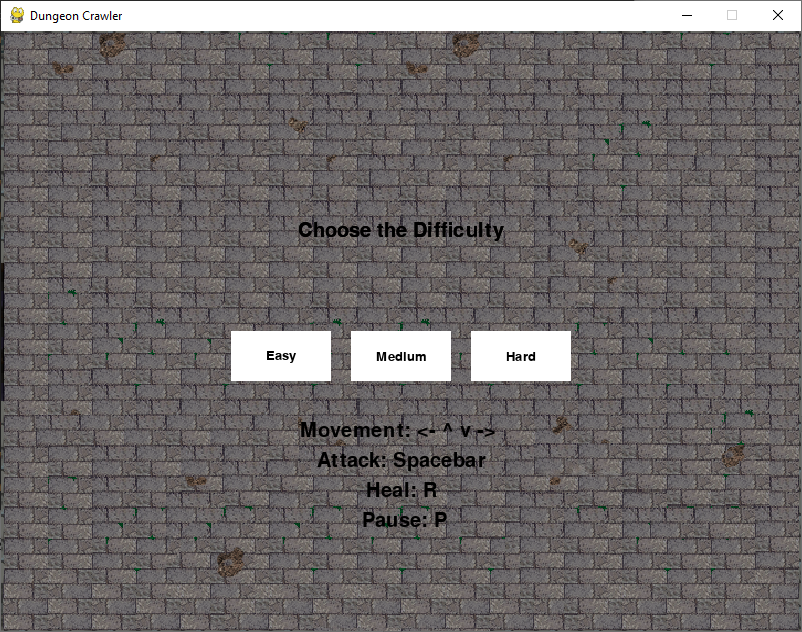
* Rooms transition was challenging as it had to keep player current stats.
* Spawning enemies had to be implemented correctly so it did not affect the game’s overall performance.

Player Movement and in-game Mechanics:

* Collision between all entities was challenging to implement.
* Changing player speed through powerups

Demo Captures

Starting screen, which include instructions and difficulty selection:



First Room, enemies and a chest can be seen. In the corner current number of Heals, Coins and Power is displayed:

A screenshot of a video game

Description automatically generated

Timed event. The player has the option to choose the Vial, throw it away, or save it for later:

A screenshot of a video game

Description automatically generated

Final room with the boss:

A screenshot of a video game

Description automatically generated

Princess spawned after beating the boss, touching it will end the game:

A screenshot of a video game

Description automatically generated

Pause screen, which can be activated by pressing P:

A screenshot of a computer

Description automatically generated

Map and game running at same time, showing multiprocessing capabilities and communication between processes:

A screenshot of a computer

Description automatically generated

End of the game, and final display of the performance in the Map process. In here it can be shown that CPU usage almost tripled throughout the duration of the game which implies the heavy processing happening. This also caused us to change directions for the implementation and remove some feature as it was crashing on the Raspberry Pi:

A screenshot of a computer

Description automatically generated

Possible Improvements and Conclusion

Improvements:

* An actual game engine instead of using pygame would improve the game vastly and development time.
* Adding animations for attacks and movements
* Enhancing performance on Raspberry pi or using different hardware.
* Scalability, making changes became harder in the latest steps of the implementation.

Conclusion:

Our team implemented several class topics using a text-based dynamic game based in Python, which could be run on a Raspberry Pi. In the limited time frame that we had to create the game we were able to implement key concepts such as multithreading and multiprocessing using the game and game map. Since we implemented everything in python so that it would run on the Raspberry pi, it made coding the game more difficult. Implementing the game easier, more in class ideas, and different aspects of the game would have been possible using a real game engine. In conclusion, with the limited time frame that our team had I believe we were able to implement some key ideas from the class in a fun interactive game on a Raspberry pi.

References:

* <https://www.pygame.org/docs/>
* <http://blog.cravenfamily.com/2009/03/running-into-walls-with-python-and.html>
* <https://api.arcade.academy/en/2.6.1/examples/sprite_rooms.html>
* <https://realpython.com/pygame-a-primer/>
* <https://realpython.com/intro-to-python-threading/>

Individuals Times spent on Project:

|  |  |
| --- | --- |
| Team Member: | Time spent/week: |
| Ignacio Gonzalez | 10-15 hours |
| Timothy Wood | 8-12 hours |
| Andrew Garcia | 8-12 hours |