The A-maze-ing Race!

Leon Lam, Max Wei, & Emily Yeh

Project Overview

In this gripping game, you take the role of the nameless protagonist, a silent white square trapped in a maze with 24 rooms. The only escape is finding the golden key and using it to unlock the wooden dungeon door. However, loose in the dungeon are green monsters, who always know your location. As you strategically race through the dungeon, they follow-- but thank goodness, their sense of smell is offset by their lack of common sense: they occasionally get themselves stuck. It's not always possible to win, but you do your best anyway.

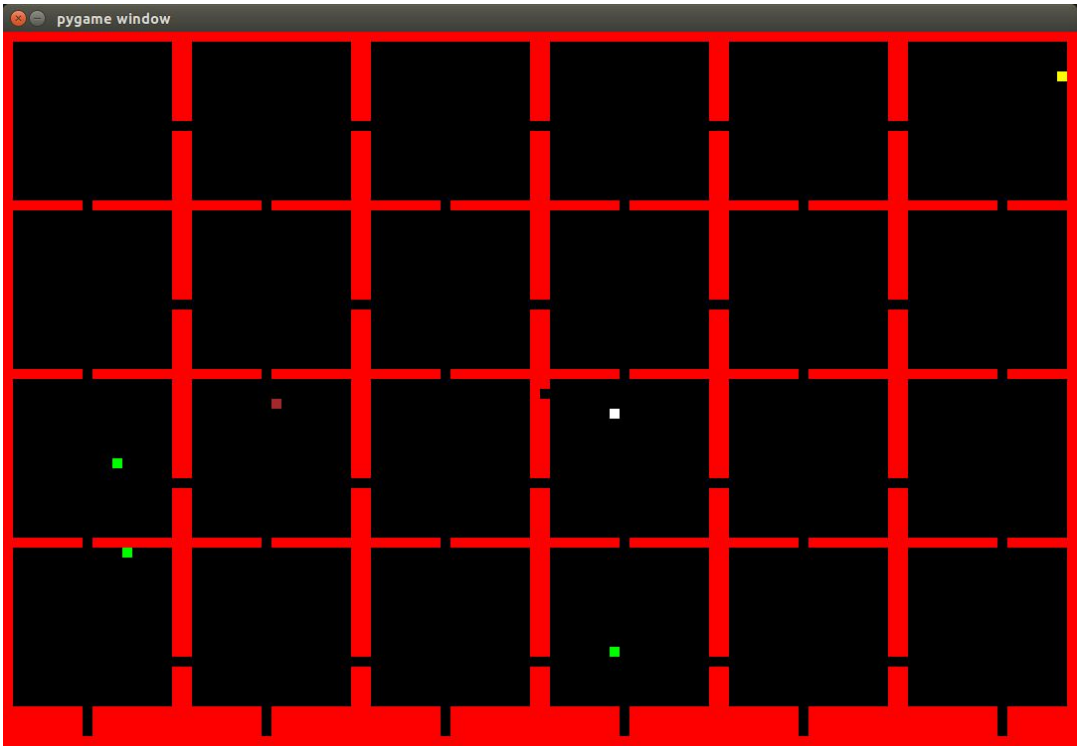
Results

We created a game that allows the player (the white square) to move through rooms and "pick up" the key (the yellow square) by moving over it, then "exit" through the wooden dungeon door (the brown square), again, by moving over it. All the while, the monsters (the green squares, of which variable numbers can spawn) will target the player and attempt (emphasis on attempt) to take the shortest path towards the player.

The game is based on a grid system. Thus, the player controls the Silent Nameless Protagonist (™ pending) through usage of the arrow keys. The up arrow key causes the Silent Nameless Protagonist to move upwards one unit. Similarly, every other directional key moves the player one unit in its traditional corresponding direction. The player cannot move through the red walls, and thus is bound within the grid through the red wall boundaries. The player also cannot pass through the door until retrieving the key.

If the player is "eaten" by a monster before they can retrieve the key and exit the dungeon door, the screen fills with red to signal gameover. However, if the player successfully nabs the key and leaves the dungeon, the screen fills with pink to signal success! Below are some screenshots of the game. The first is a screenshot of the game when it has just started; the second is a screenshot of the game when the player is eaten by a monster; and the third is a screenshot of the game when the player successfully retrieves the key and leaves the dungeon.

Screenshot 1: Typical Game Screen

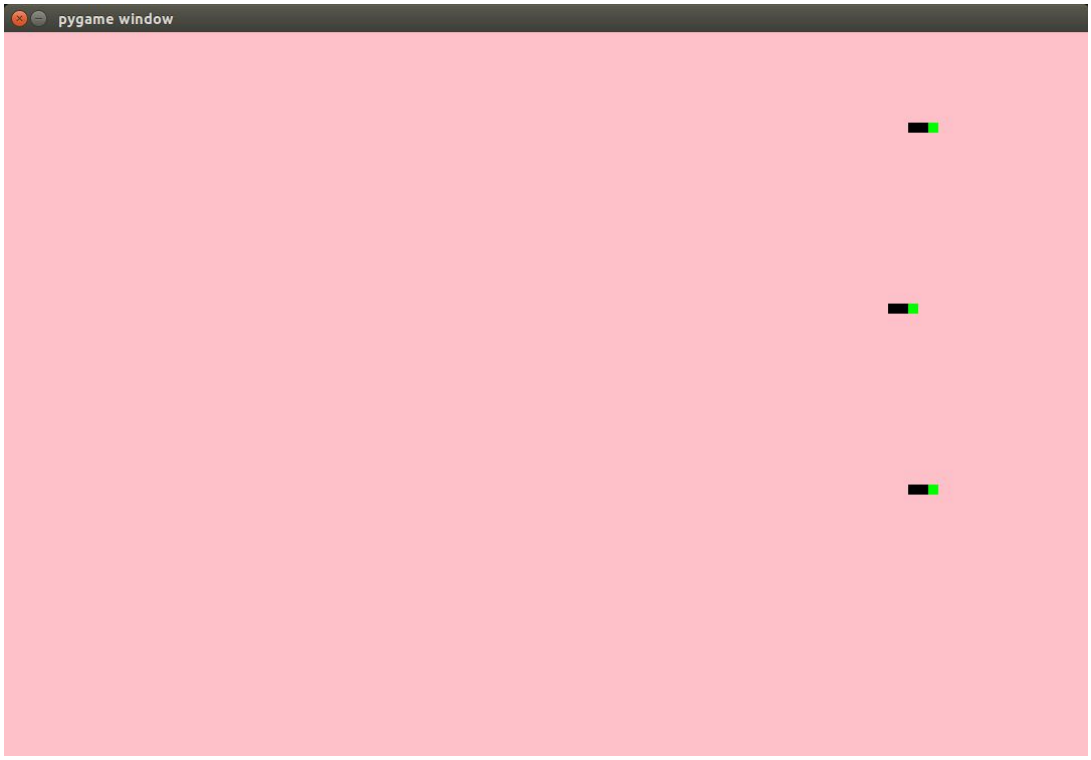


An example of the screen when the game has just started. Aren't those some stunning retro graphics?

Screenshot 2: Losing Game Screen

An example of the screen when the player is eaten by a monster. It's a little horrifying.

Screenshot 3: Winning Game Screen

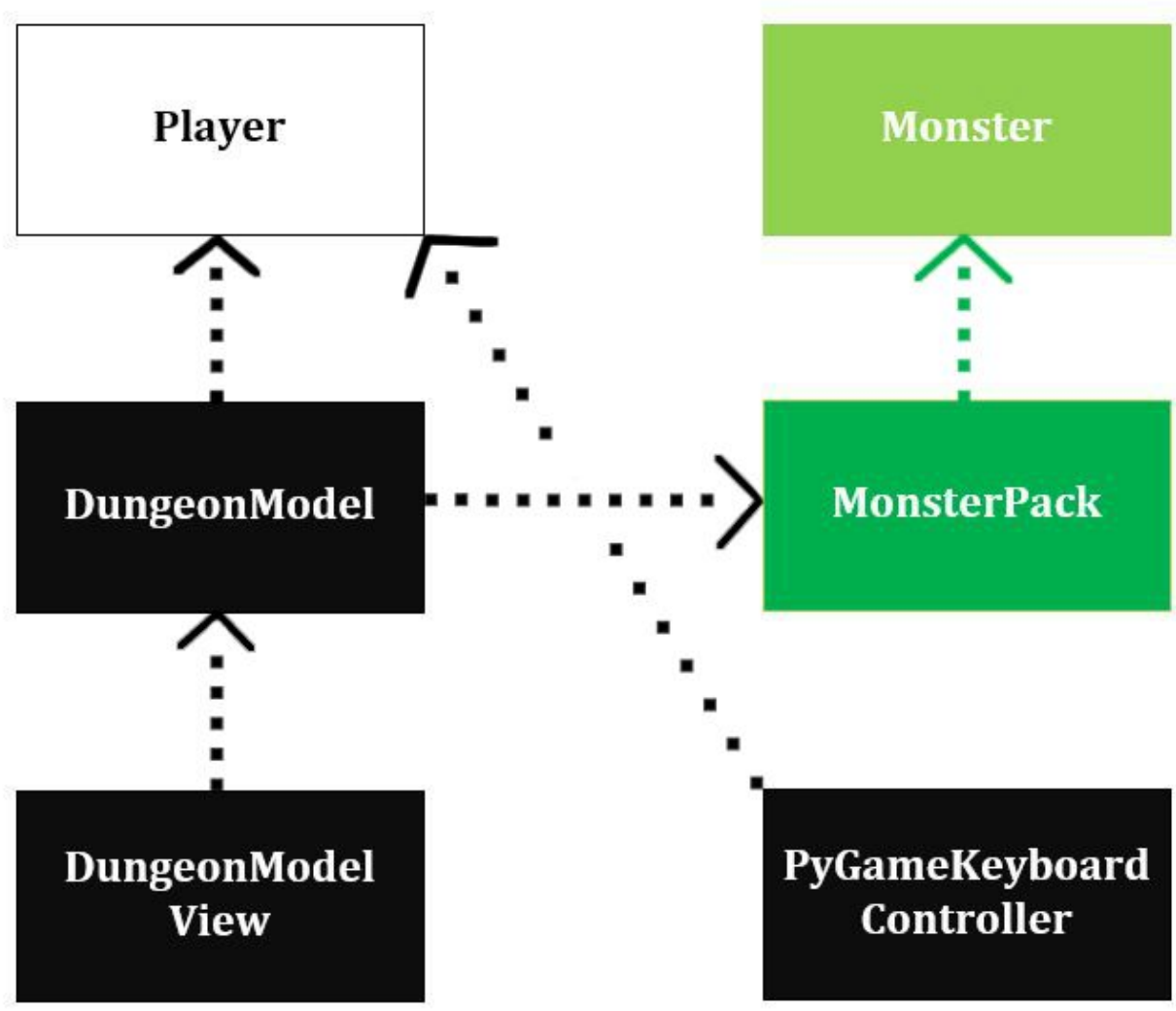


An example of the screen when the player wins the game. Pepto-Bismol pink for the win!

Implementation

Below is our UML class diagram. There were six classes: Player, Monster, MonsterPack, DungeonModel, DungeonModelView, and PyGameKeyboardController. The names are fairly self-explanatory. In this mini-project, we did not make much use of inheritance between the classes; as such, there are no "is a" relationships. Instead, the following list provides details about how each class relates to others:

- ★ MonsterPack has a (list consisting of) Monster.
- ★ DungeonModel has a Player and a MonsterPack (which has a [list consisting of] Monster).
- ★ DungeonModelView has a DungeonModel (which has a Player and a MonsterPack [which has a (list consisting of) Monster]).
- ★ PyGameKeyboardController has a Player.



We decided to use a grid system for the world-- that way, the player and monsters are guaranteed to move at the same speed. Generating an array with numpy, we populated it with impassable walls,, a key, a door, monsters, and the player. The walls are given values of 1, which is the "unpassable" value. All other spaces are given values of 0, which is the "passable" value. A

monster or a player can enter an adjacent space if it has a value of 0. In contrast, if the adjacent space has a value of 1, the monster or player can not enter the space. We decided to make the walls part of a fixed map, for the sake of getting things to work; initially, we had plans to make the map a maze, and we did briefly work on room generation algorithms. However, we eventually decided that generating a map may be beyond our capabilities at this point.

The idea behind the monster AI is that the monster must attempt to align their x-position with the player, then their y-position. If they bump into a corner, they'll hug the wall until they reach an opening, then resume their movement. They wall crawl through recognizing that they are not making progress towards the player when they corner themselves, then start executing a modified behavior set that blocks them from attempting to move further in a stuck direction through placing temporary "monster only unpassable" blocks.

Reflection

In this project, there were many things that went well. Our team dynamic was great-- we were supportive of each other and we always met when we agreed to meet. We didn't really divide up the work, but simply spent time working on everything together. We also managed to cobble together a working game, which was pretty exciting! None of us have much experience with programming in Python, and none of us had any experience with developing games with Python prior to this mini-project. In fact, this mini-project was so interesting, that we plan to continue working on it, even after we have submitted it.

Although many things went well, there were definitely still areas of possible improvement. For example, our game is not very pretty, since we threw it together using basic shapes. We had originally hoped to use sprites and backgrounds to make our game more realistic and interesting, but this proved to be beyond our current capabilities. Furthermore, the initial design of the game was to form a roguelike dungeon crawler in which the player could go through a one screen map of randomly generated rooms. However, due to time constraints, we decided to start off with a semi-complex pre-determined map to further testing. Eventually, this developed into the main game idea of more strategy. The monster AI further contributed to the strategy aspect by having the player take into account the current map conditions and thus the best way to abuse the monster AI. Overall, we think that the end project was scoped appropriately for the time constraints given, in that we managed to create a working, playable game with a clear and accomplishable goal.

Although we were not able to incorporate all the original ideas from the brainstorm, the project allowed us to gain experience in the Python structure and logic behind coding an interactive project. In the future, we hope to continue adding more complex elements to the game, such as randomly generated rooms, in order to improve on the project and on our skills as a whole. It would have helped to have had more experience with Pygame prior to this mini-project, and in fact, next time, it would probably be a good idea to start by having everyone in our team try the AI toolbox exercise, or try creating an extremely basic game using Pygame, just to help us gain a better understanding of what it can do. In general, though, we gained a far better understanding of game development using Python while we were working on this mini-project-- and we had a lot of fun!