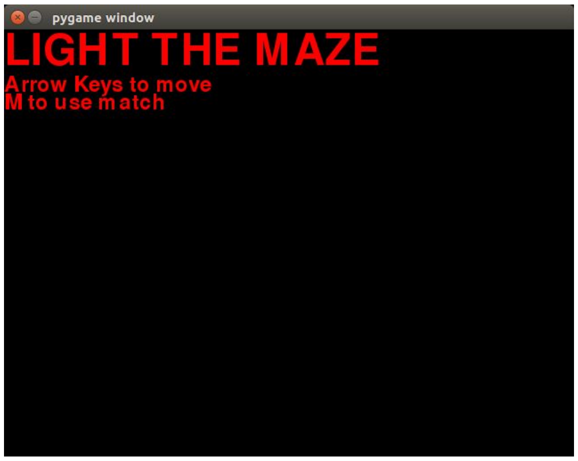
Project Overview

Light the Maze is a maze game that asks the user to pay close attention. The player is placed in a dark environment with a limited number of matches. Each match they light gives them 3 seconds to look at the map. They must then try to navigate to end of the maze using the arrow keys, but if they take a wrong turn and bump into a wall, a player receives a point, the goal then is to get to the end of the maze with the fewest points. Light the Maze currently has two levels.

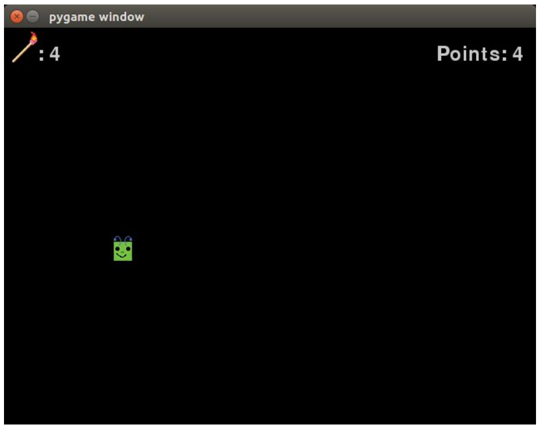
Results

The results are as expected, and the game is really exciting- if you designed it.
We have little evidence that the game is fun, if it doesn't automatically just by working fill you with pride but for our not one but 2(two) current users, it has that effect.

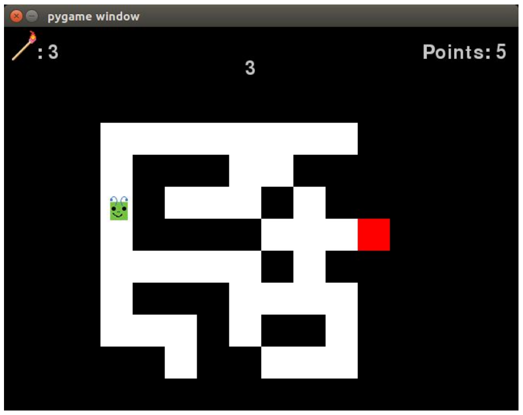


LIGHT THE MAZE
Arrow Keys to move
M to use match

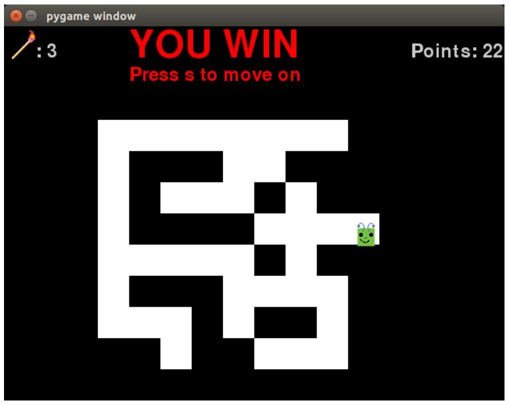
The game starts with an instruction page, that lasts a few seconds and leads into the actual play.



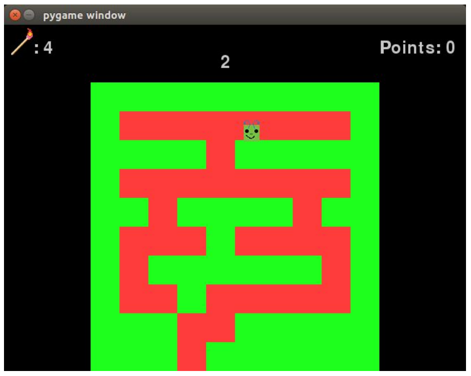
Upon entering the game the screen is black but for the player. The user can move the character in any direction unless they are moving into a wall, that's how to earn points(bad) the number of points and matches left are listed at the top of the screen.



A user can light a match by hitting the M key, a timer is then started that counts down 3 seconds. During these seconds the player is immobilized and the map is displayed.



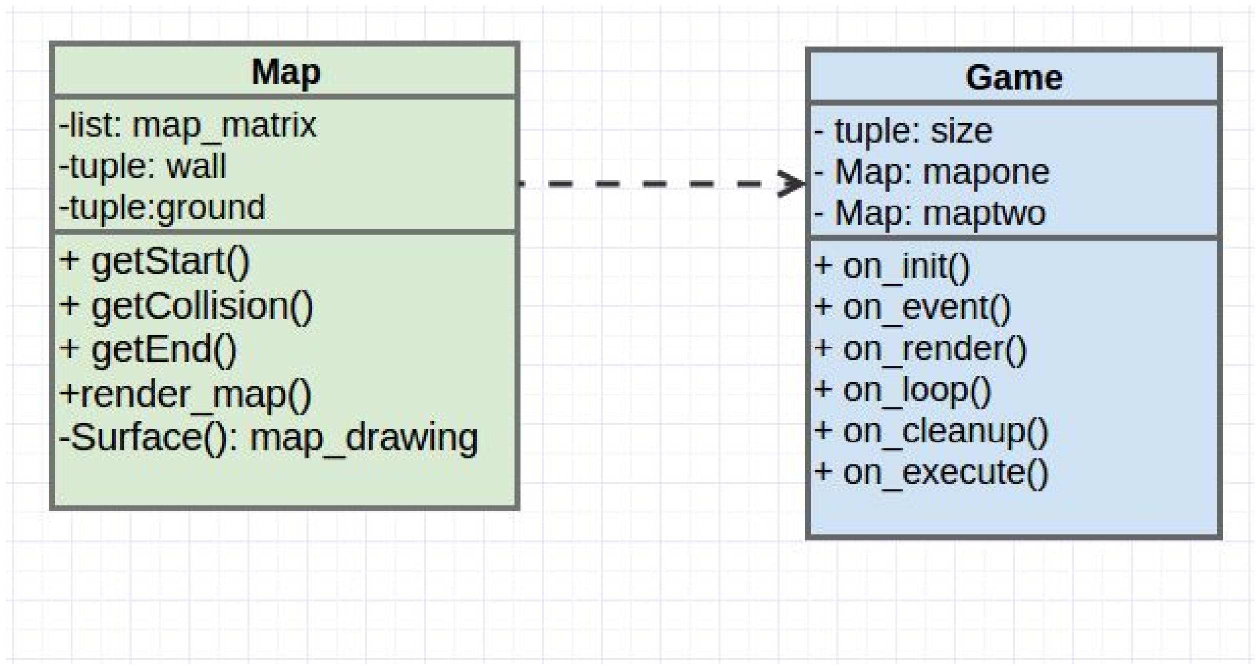
Once a player successfully navigates to to the end of the maze they are brought to the winning screen, immobilized and given the option to move on.



If the player moves on they are presented with the next level.

Implementation *[~2-3 paragraphs + UML diagram]*
Describe your implementation at a system architecture level. Include a UML class diagram, and talk about the major components, algorithms, data structures and how they fit together. You

should also discuss at least one design decision where you had to choose between multiple alternatives, and explain why you made the choice you did.



This is the basic structure of our game. We built a game class that has 6 main methods, seen above. They each handle a different part of the game: on_init creates game variables, on_event handles input, on_render displays game objects, on_loop does game math, on_cleanup shuts the game down, and on_execute runs everything. This breaks up our code into easily manageable chunks. It lets us better organize all of the function calls so our code is more readable.

The Map object is the key to the entire game working. It takes input from a separate python definition file where we define the map matrix (Describes the layout of the map) and wall and ground colors. Inside the map are a few methods. They let us call the start position, end position, and check squares for collisions. The render_map method is entirely internal, creating a surface-object attribute, map_drawing, that the game class can call when it needs to render the map. The render_map method handles the logic of laying out each block.

The Game has many different ways of using the map. When the player moves, the map object checks the block they are moving into for a wall. If a wall exists there, they do not move and get a point. If not, they are allowed to move. It also dictates where the player starts and ends. When a player moves, the game calls the Map.getEnd() method to check to see if they are moving into the end of the map. If they are, the game throws a flag, stops all movement, and waits for the player to hit “s” to restart. On restart, the map changes.

We implemented a timer in pygame for the matches that light up the maze. When a match is used, the game throws a flag. In the on_loop() method, it checks the time since that flag was thrown. If it has been less than three seconds, the game displays the map and does not allow the player to move. Otherwise, the game resets the flag, and allows the player to move again.

Reflection

All in all the design and building of this game went really well! This was in large part due to frequent communication, working in the same place, not moving on until both of us were on the same page and starting with a solid and clear framework. Our project started of WAY out of scope but was quickly corralled into a much more possible goal. This went better/faster than expected, so we were able to focus on getting it to be just how we wanted , which was definitely a plus. We started with a loose framework and tested frequently each time we added something so nothing was too hard to diagnose.

The process was not harried, and stayed pretty fun! We had minimal conflict, but what drama was had was pretty easy to resolve with communication/a little distance/ we have been working on this too long. Whenever we were working, we were in the same room, even if not pair programing. This ensured that our individual contributions fit together seamlessly and that we both understood every aspect of the code.