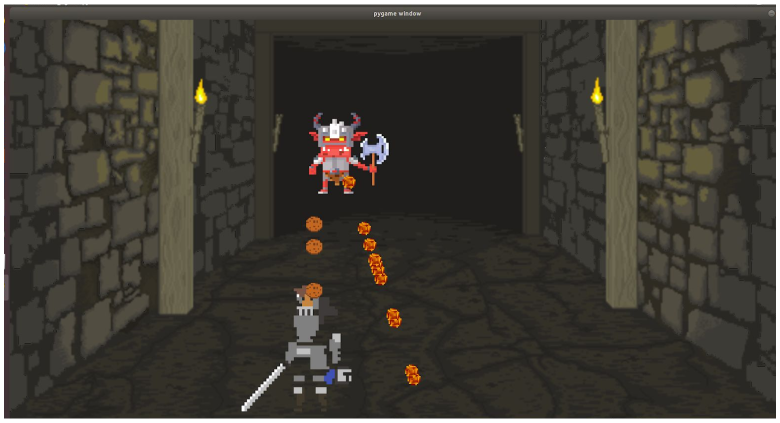
Project Overview

In this project, we used pygame collision detection, sprites, and blit'ed images to build a dungeon game where a knight shoots arrows or cookies in order to slay or befriend a monster while avoiding the monster's fireballs. The knight's collision with a fireball or the monster's collision with an arrow cause a decrease in health. When the monster's health runs out, or if it is fed enough cookies, or if the monster slays the knight, then the game ends.

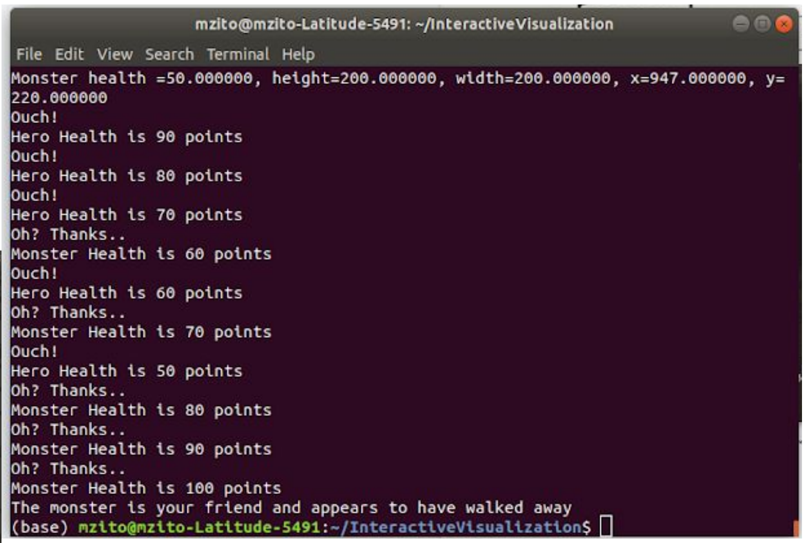
Results

The hero is a pygame sprite with an image and a rectangle as attributes. The image is uploaded from our directory. The hero moves based on a controller that tracks the mouse. Left clicking fires an arrow upward towards the monster, while right clicking fires a cookie upward at the monster. The arrows decrease the health of the monster, and once the monster's health is at 0, it dies and ends the game. The cookies increase the health of the monster, and once the monster's health is double its original, it leaves which ends the game. The hero can also die if it is hit by enough fireballs, which is the last game ending option.

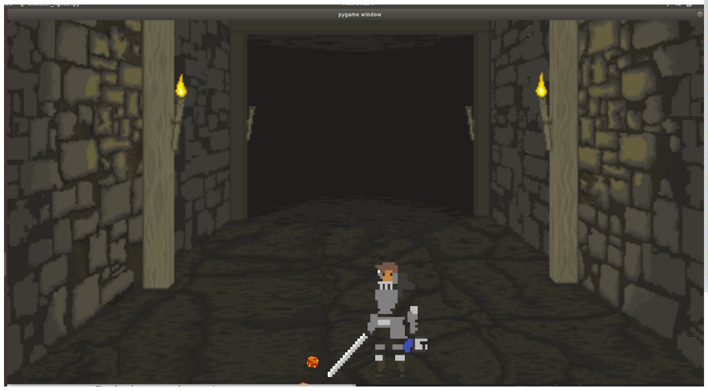


The monster moves back and forth between the walls of the dungeon with a constant velocity. The dungeon narrows the higher up on the screen you are, so the monster only moves across about half of the screen. This leaves the far left and far right of the screen for the knight to hide from fireballs and allows players to get a feel for the game before jumping into the action.

The terminal displays changes in the health of the monster and the hero along with fun little messages. If the monster gets enough cookies then it is your friend and walks away. The terminal to the right and the visual below display this ending.

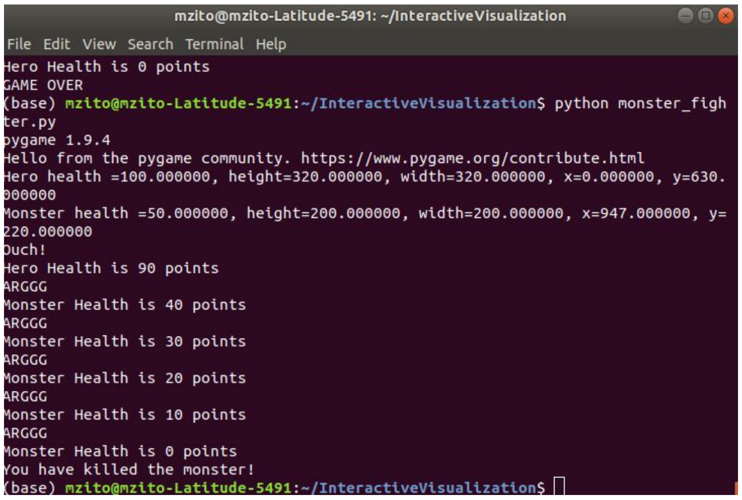


```
mzito@mzito-Latitude-5491: ~/InteractiveVisualization
Monster health =50.000000, height=200.000000, width=200.000000, x=947.000000, y=
220.000000
Ouch!
Hero Health is 90 points
Ouch!
Hero Health is 80 points
Ouch!
Hero Health is 70 points
Oh? Thanks..
Monster Health is 60 points
Ouch!
Hero Health is 60 points
Oh? Thanks..
Monster Health is 70 points
Ouch!
Hero Health is 50 points
Oh? Thanks..
Monster Health is 80 points
Oh? Thanks..
Monster Health is 90 points
Oh? Thanks..
Monster Health is 100 points
The monster is your friend and appears to have walked away
(base) mzito@mzito-Latitude-5491:~/InteractiveVisualization$
```

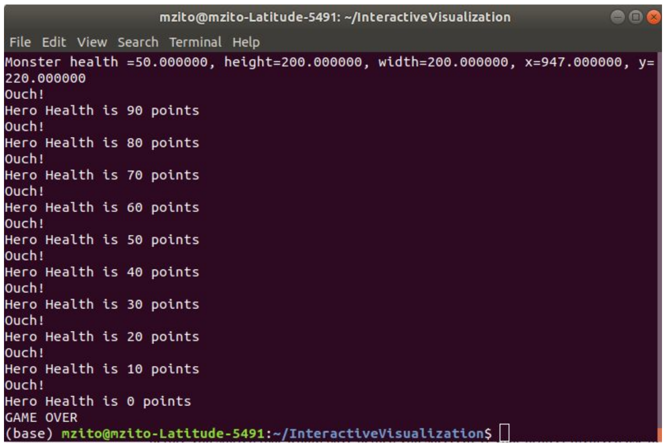


Killing the monster displays the same visual, with the monster gone and the knight still standing, but the terminal returns different messaging, which can be seen to the right.

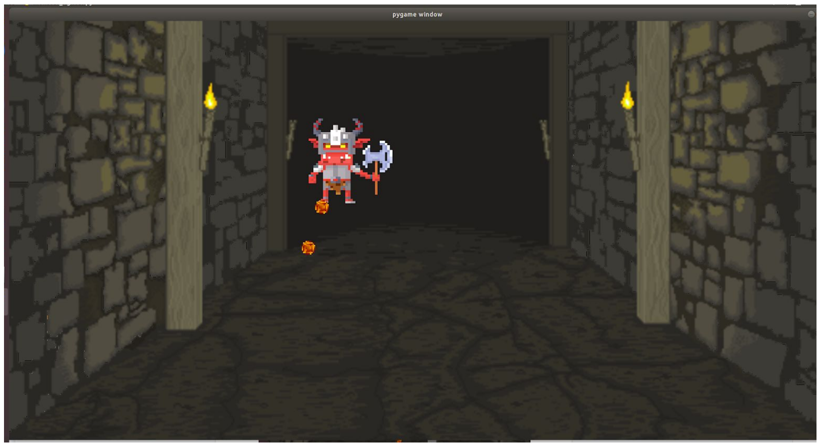
If the monster defeats the knight by decreasing the knights health to 0 with fireball collisions, then the visual and the terminal below are displayed.


mzito@mzito-Latitude-5491: ~/InteractiveVisualization

```
Hero Health is 0 points
GAME OVER
(base) mzito@mzito-Latitude-5491:~/InteractiveVisualization$ python monster_figh
ter.py
pygame 1.9.4
Hello from the pygame community. https://www.pygame.org/contribute.html
Hero health =100.000000, height=320.000000, width=320.000000, x=0.000000, y=630.
000000
Monster health =50.000000, height=200.000000, width=200.000000, x=947.000000, y=
220.000000
Ouch!
Hero Health is 90 points
ARGGG
Monster Health is 40 points
ARGGG
Monster Health is 30 points
ARGGG
Monster Health is 20 points
ARGGG
Monster Health is 10 points
ARGGG
Monster Health is 0 points
You have killed the monster!
(base) mzito@mzito-Latitude-5491:~/InteractiveVisualization$
```


mzito@mzito-Latitude-5491: ~/InteractiveVisualization

```
Monster health =50.000000, height=200.000000, width=200.000000, x=947.000000, y=
220.000000
Ouch!
Hero Health is 90 points
Ouch!
Hero Health is 80 points
Ouch!
Hero Health is 70 points
Ouch!
Hero Health is 60 points
Ouch!
Hero Health is 50 points
Ouch!
Hero Health is 40 points
Ouch!
Hero Health is 30 points
Ouch!
Hero Health is 20 points
Ouch!
Hero Health is 10 points
Ouch!
Hero Health is 0 points
GAME OVER
(base) mzito@mzito-Latitude-5491:~/InteractiveVisualization$
```



Implementation

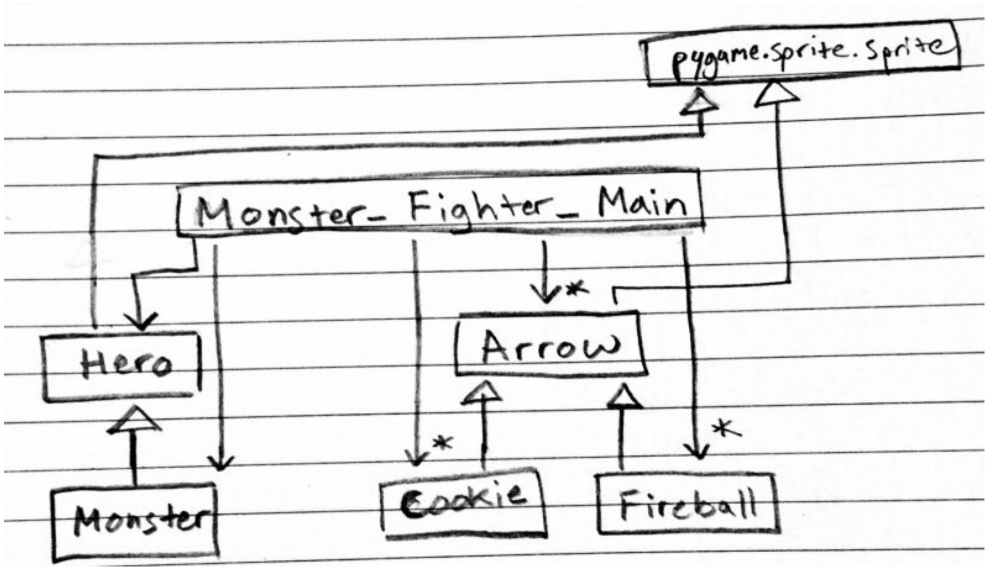
The main class in our program is Monster_Fighter_Main, which: encodes a display of the game; initializes a monster and knight; includes the functions to create arrows and cookies and add them to their respective sprite classes; includes the functions to update, load, and add to our sprite groups of arrows and cookies; includes a function to check if the conditions to end the game are met; and finally has a MainLoop function that initializes the game with the hero and monster and then loops every .001 seconds to check if the mouse has moved or clicked or if the game has ended.

The hero class is a pygame sprite and is for our knight and monster. A load_image() helper function which returns an image and a rectangle helps us load an image with the same name as the hero ("Hero" or "Monster"). Heros have an image attribute and a rectangle attribute that work well with the sprite functionality to help us display a moving image and test for hitboxes. The x and y attribute of our hero class correspond to the left and top of our hero's rectangle, respectively. The lower_health function within the hero class helps us keep track of health and the update function tests for a collision with another sprite. If there is a collision, the lower_health function is called.

Monster is defined as a hero and therefore inherits all of its attributes and functions. Because the fireballs have to shoot automatically and the monster has to move automatically, the hero update function is overridden in order to implement a constant velocity that reverses directions at certain coordinates. Fireballs are shot at random times.

Arrows are simple sprites that move with a constant y velocity. The update function simply moves the arrow's rectangle. Fireball and cookie are arrows and therefore inherit their attributes. We have 3 different sprite groups for the 3 different projectiles because of their different movement, damage, and collisions.

A problem that we encountered was whether we should make arrows into sprites. We would have to go back and completely change a lot of our work in order to do that, but doing so would save us the work of creating our own collision detection function. We looked into how to write collision detection and it was super complicated, so we decided to go back and make our arrows into sprites and it is definitely worth it. Pygame has sprites integrated for a reason and we should definitely use them! We learned about using all of the tools at our disposal!



Reflection

From a process point of view, what went well was working on the code together, bouncing ideas off of each other, and fixing problems together. What didn't work well was jumping right into implementation without completely going through tutorials and understanding what we were doing. Because we didn't do enough research before we started, we didn't realize how useful sprites were and started implementing classes that weren't sprites.

In addition, we used Ben's framework from his brick breaker game example as the framework for our game and I don't think it worked very well because we were using sprites and the way that sprites have to update is completely different from Ben's framework. We struggled implementing within Ben's framework for a while before finally deciding that we needed to build our own from scratch. Transitioning to a new framework that had our sprite update functions within a Monster_Fighter_Main class and didn't differentiate between view and model really helped us as we moved forward. This could have been avoided with better planning and design before we jumped right in, again emphasizing our need to take our time at the beginning of projects.

Our scope was reasonable. I think that if we didn't run into so many problems, then we would have had the opportunity to implement many of our reach goals. Our team worked great together and we really had a lot of fun.