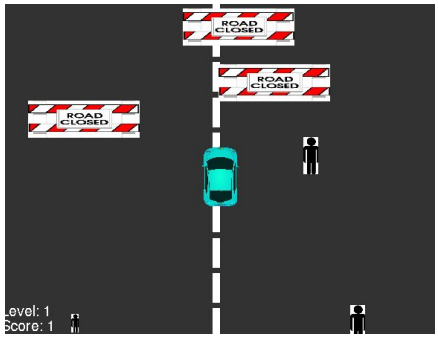
Bryce and Jamie's Project Write-up and Reflection

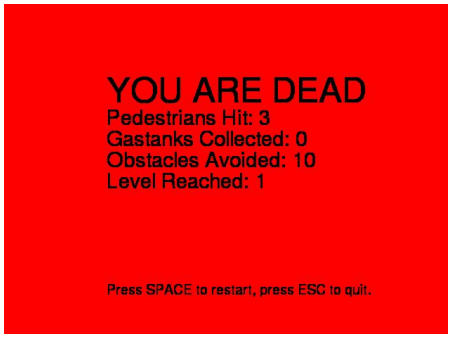
Project Overview

We designed a video game similar to some existing games, such as Temple Run or Subway Surfer, called Road Rage. The objective of the game is to drive the car, avoiding obstacles, collecting gas tanks as to not run out of gas, and hitting pedestrians for points. The game is exciting because the obstacles, gas tanks, and pedestrians are all randomly generated, and there are multiple levels that vary the speed of both the environment objects and the car itself.

Results

We produced a video game that tracked points, level, gas levels, and high scores. The following images are images of our start screen, our kill screen, and the game itself. The start screen displays the rules of the game as well as the top high scores. The kill screen displays the player's number of pedestrians hit, gas tanks collected, obstacles avoided, and levels reached. During the game, the player can maneuver the car in four directions using arrow keys, and pedestrians and gas tanks disappear once hit or collected by the car, respectively.



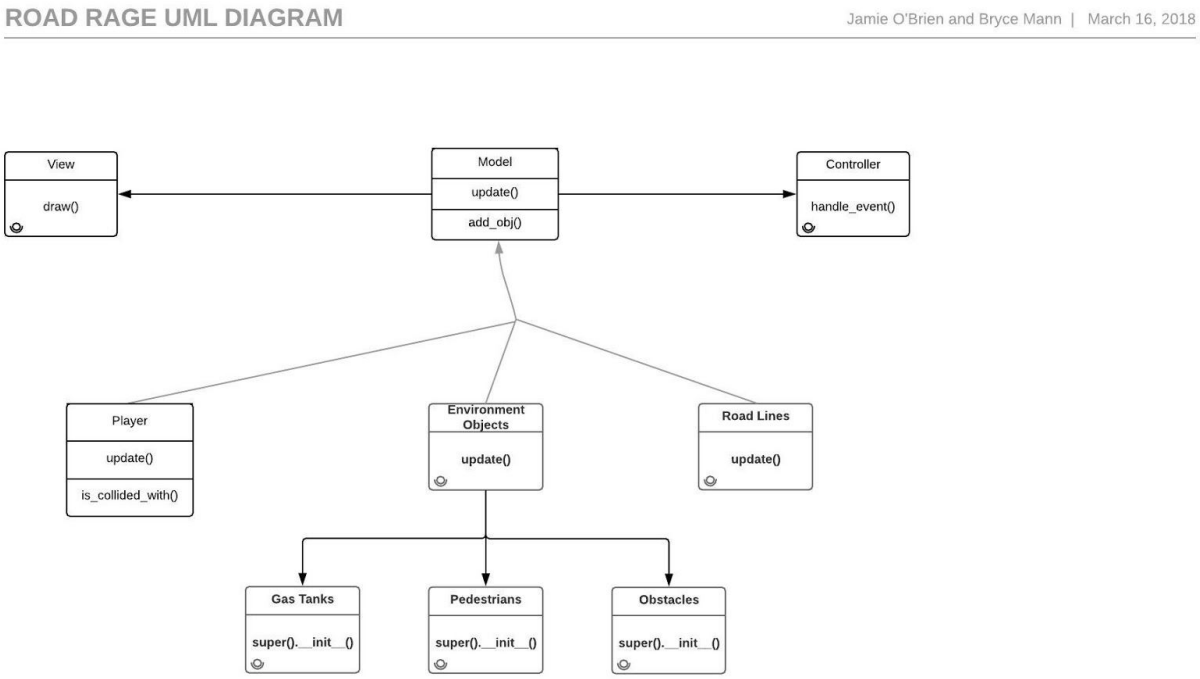


From play testing this game on our peers, we found that the game is challenging yet exciting, fun, and somewhat addictive. The highest score achieved by a peer so far is 43, at level 2. All of the reviews we've gotten from peers have been positive, one of them being "This could be the new flappy bird!"

Implementation

To create this game we used an MVC framework in order to have easily modified code. Our model class was responsible for keeping track of the user controlled player, randomly adding objects to the road for the player to interact with, detecting collisions between the objects and the player, and eliminating the objects once they moved off screen; our view class took this model and drew the updated figures on screen every cycle, and our controller class modified the player class within the model so that it could be maneuvered. Within the death and main menu screens we also utilized the pickle module to be able to store and display the top 3 local high scores.

While we researched some of the ways to make games in python, and pygame in particular, we came to the decision to have our player and objects inherit from the pygame Sprite class. Instead of having to define the areas that the player and objects were taking up and detect intersections (collisions) manually, which would not have been the most interesting task requiring math and trial and error, we were able to use the built in attributes and methods to take care of it and focus our efforts on refining the game mechanics and adding more interesting aspects to the game. Being able to use the pygame Group structure, an easy way to organize and keep track of Sprites, was just an added bonus that made our code a lot easier to write, understand, and modify.



Reflection

When we started this project, we had intended to make a three-dimensional game,with turns and jumps, much more like Temple Run, and our stretch goal was to make a randomly generated map, drawing from a handful of predefined building blocks. As we tried to figure out the three dimensional aspects, we experienced trouble comparing indices of matrices and arrays, and consequently pivoted to a two-dimensional version of the game. However, we still did achieve our goal of a randomly generated map, but given predefined environment objects. The two-dimensional version was much more appropriately scoped for the length of the project. If we'd had more time to work on the game, we would work out the math to ensure that there isn't a possibility of the whole road being blocked by obstacles, and come up with more ways to make the game challenging.

When testing our code, we agreed that using the pygame display window to gauge whether our code was producing the results we expected was more effective than unit testing, and consequently didn't include unit tests in our code. We worked well when together, and also

handled individual work and collaboration through git really well. Scheduling was a bit difficult at time, but we made up for it with individual efforts. We split the work fairly evenly and helped each other out when one of us got stuck. Overall, we worked well together and produced the game smoothly and efficiently.