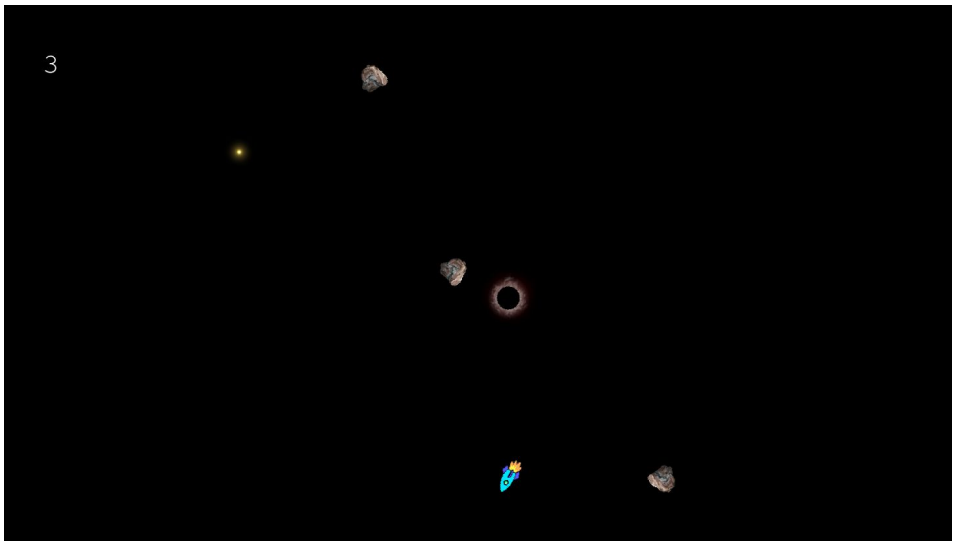
Project Overview:

For our project, we created a video game with more realistic space physics such as gravity, conservation of energy, and black hole related time dilation.

Results:

Black Hole Simulator 2017 was a success. The player acts as a spaceship navigating around a black hole collecting small yellow energy capsules and avoiding space debris. We implemented a time dilation aspect, causing other objects to appear faster as the player approaches the black hole and slower as they approach it.



The game has realistic gravity and orbits, somewhat less-realistic collision between asteroids, and a method for keeping track of the player's score. The player controls the ship using keyboard input.

Implementation:

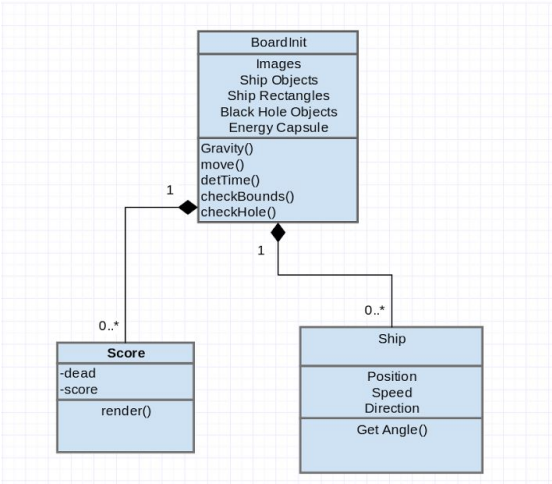
Our program was rendered in pygame. It relies mainly on a list of objects from the Ship class, the first of which is a ship that the player controls while the rest are rendered as asteroids. Every 33 milliseconds the program will read user input such as ship rotation or thrust and execute accordingly. Additionally, it will loop through all the objects in the ship list and execute the following functions:

gravity - updates speed according to the gravity acting on the object

move - moves the object according to its speed

checkbounds - checks if ship is off screen
checkHole - checks if the object enters the Black Hole
checkDot - checks if the ship collects the energy capsule

Each of these function will also implement helper functions such as distance() which will return the distance of the object to the black hole or detTime() which will determine the required time dilation for the player and asteroids based on distance from the Black Hole.



We added the support classes of Ship and Score. Ship stores the necessary information for the ship and each asteroid such as position, speed, angle from vertical. The Score class keeps track of how many Energy Capsules were collected and whether or not the player has been hit or sucked into the black hole.

Instead of simply rotating the image when rotate is called. We decided to re-import the image for the ship/asteroid each time the image is rotated and instead keep track of its angle from vertical and rotate the image to that angle. The reason for that is the image loses quality each time it is rotate, so by re-importing each time, there is no noticeable decrease in quality.

Reflection:

Our team worked well together, and work was distributed fairly between members. We were both enthusiastic and willing to work on the project beyond a minimum viable project. An improvement would be the reorganization of the code; it is, for the most part, clustered into a single class BoardInit. To increase readability and organization, we could have separated it into several more classes as well as had further separation between the control, frame, and display aspects of the game.

In general, we split the work where it was beneficial and practical. One of us wanted more practice with classes, methods, and object-oriented programming, so they did the peripheral scripts for the game, while the other worked primarily on the main script. We coordinated our pushes and pulls such that we avoided merge conflicts, which was made easier by the fact that a lot of our contributions were made in the same room together. The only issue we had as a team was an indecisiveness in which project to pursue; we originally wanted to do something involving vision or image processing rather than a game. In hindsight, something more complex could have been more beneficial, but was more difficult to find proper support for.