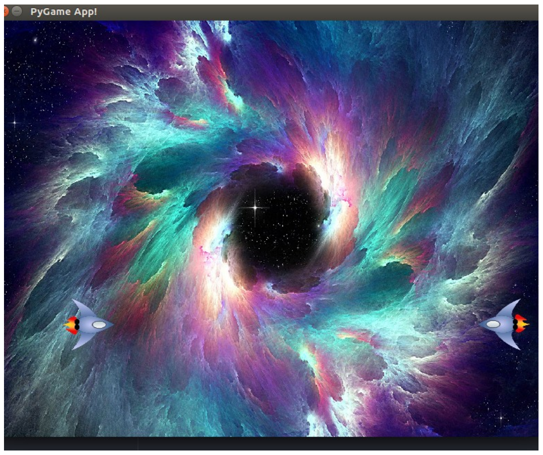
Mini Project 4 – Interactive Programming
Anil Patel and Nathan Lepore

Project Overview

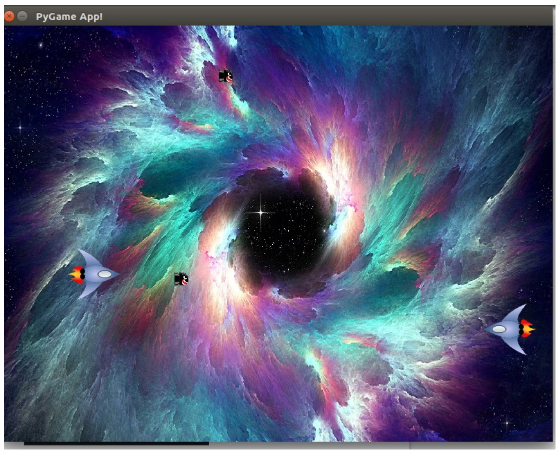
Our project is an arcade-style game developed in pygame to be played by two players on one computer. Our work consisted of creating a development plan, researching and understanding the mechanics of pygame, and finally implementing and revising the game code together.

Results

We managed to fully create a first-pass model of the game in pygame. Upon startup this is the gameplay platform visually.



We were able to create player movement fairly easily. However, to move forward with other parts of the game including movement by holding buttons down, shooting bullets, and collisions, our high-level code structure had to be revised multiple times. Each revision accounted for something new we wanted to add to the game or us gaining a better understanding of python and pygame.



The most difficult task was creating bullets that shot from players. Challenges we encountered including tracking multiple bullets, tracking bullets from separate players, accounting for collision mechanics with walls and with players, and creating velocities/angles.

Concluding steps were to translate player/bullet collisions into transitioning to and end-screen, which would end the game.

Implementation

We implemented our project with two main classes, a player class, and a bullet class that interfaced with the player class. This interfacing allowed for collisions with players and for different players to shoot in different directions. Our model is based in pygame. On a high level we gather all keys pressed at every screen refresh and use the events functionality in pygame to udpate our sprites appropriately. This includes player movements and bullet initiations. Our bullets are controlled in a loop that tracks all of the bullet locations and checks for collisions with walls and players. Our view component gathers all the sprites with their updates locations and calls on the draw methods for each of them. Collisions with players ends the model and loads the game over screen.

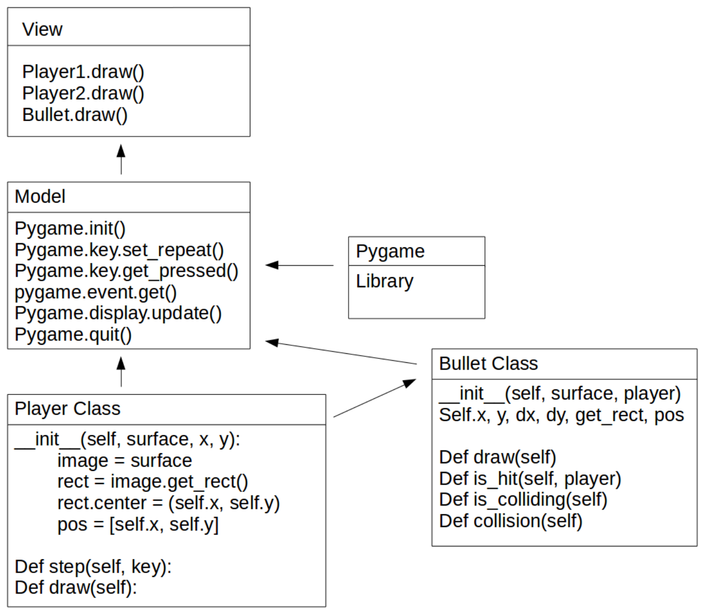
One decision we made was to gather all keys pressed at one instance and then use that data to update all sprites at the same time and display them. Before that we had our events section update the model as keyboard presses happened, which caused sprites not to be able to move at the same time.

Reflection

In hindsight we needed a lot more research into the mechanics of pygame before we started coding. We did a very good job preparing a plan for testing incrementally but the way we implemented it didn't work super effectively. We followed our plan for incrementally adding components and functionality to the game but we didn't fulfill early components of the dev plan in a way that was conducive for future components. For example, we had a rudimentary setup for moving a single character around the screen, but when we wanted to move forward with multiple players, we had to restructure our movement functions so that multiple sprites could be moving at the same time.

Also we missed our mid-project check-in, which is reflective of the fact that we didn't prioritize this project as highly as we should have early in the project because of the converging pre-break projects hitting crunch time. We will be following up on this by choosing this project to be extended for MP5 and properly budget our time and energy to make something that fully fulfills the vision we had for our project at the onset.

In terms of teaming, we did a good job dividing work between the two of us in a way that we were working on different functions and mechanics simultaneously so that we made good use of meeting time. Additionally our use of git was decent in communicating our work but we didn't fully use the branch functionality as well as we could have.


View
Player1.draw()
Player2.draw()
Bullet.draw()
Model
Pygame.init()
Pygame.key.set_repeat()
Pygame.key.get_pressed()
pygame.event.get()
Pygame.display.update()
Pygame.quit()
Pygame
Library
Bullet Class
__init__(self, surface, player)
Self.x, y, dx, dy, get_rect, pos
Def draw(self)
Def is_hit(self, player)
Def is_colliding(self)
Def collision(self)
Player Class
__init__(self, surface, x, y):
image = surface
rect = image.get_rect()
rect.center = (self.x, self.y)
pos = [self.x, self.y]
Def step(self, key):
Def draw(self):