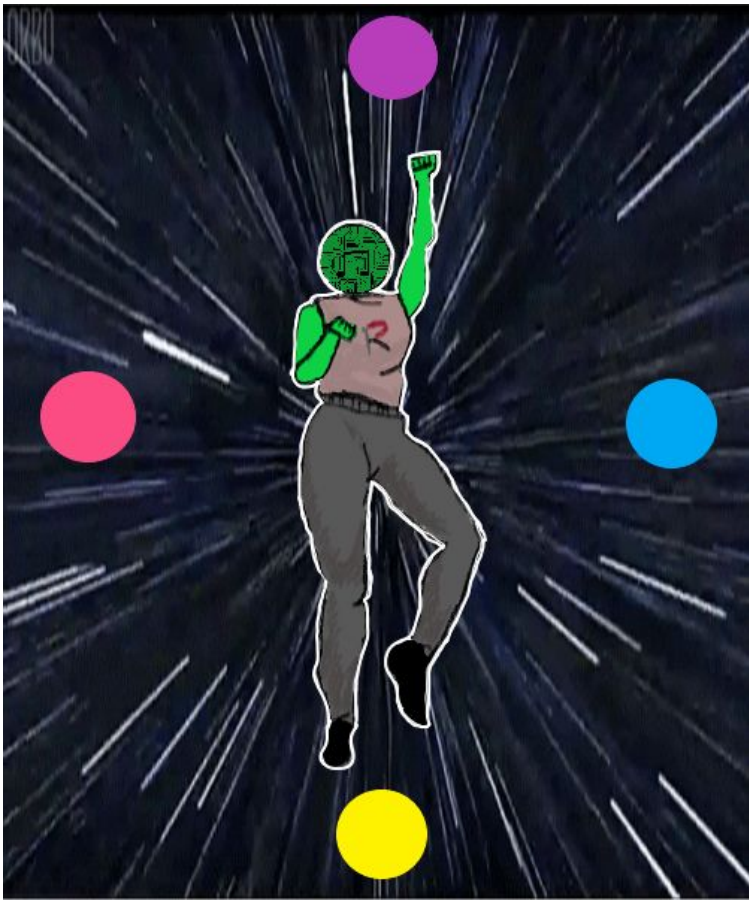
Algorymolto Vivace

Final Catapult Report - Team 09

“Beat Fighter”



Allison Albernathie, Eureka High School, Eureka, MO

Cassandra Lutes, Morristown Jr-Sr High School, Morristown, IN

Isabella Patterson, Etiwanda High School, Rancho Cucamonga, CA

7/18/2019

Summary

The goal of our project was to create a fully-functioning, playable video game using the skills we acquired during the first three days of Operation Catapult. We decided to create a rhythm-based fighting game in which the player would attack in time with the beat of the music to defeat enemies. We planned for the game to include features, such as selectable characters, a variety of songs with unique choreography, an opening screen in which the player can choose both their character and the song, a screen for winning, a screen for losing, and the ability for the player to restart the game by pressing the space bar.

To create our game, we used Python, a coding language known for being easy to read and well-suited to both small-scale and larger projects. Its simplicity and clear logic made it ideal for this project, as only one of our members had prior programming experience of any significance. We started learning how to code and use the library Pygame on Monday, July 8, completing 7 programs that taught us the essentials needed to get our game functioning. Through these 7 programs, we learned not only how to program, but also how to research whenever we don't know how to code something, as well as, how to read and fix errors in the code.

We created several classes for features we knew we would need, including the base of the player character, the enemies and their different directions, the health bar, and the selectable faces of several counselors who would become avatar characters. We began adding on features as necessary, such as a hit box for damage and music for the win and lose screens, and adjusted

according to the feedback we received from teachers and fellow students alike. We then had to work to get the game functioning which was a feat for our team, our teaching assistants, and our professor. Although it took much trial and error, we were able to complete the game within the desired time frame.

In conclusion, our team learned how to use the Python coding language to successfully complete our project. The resulting game met all our planned goals, with three playable characters, many levels with different music, opening and closing screens, and a reset button. In conclusion, we met all our goals and exceeded our expectations, creating a game that is fun, functional, and was an educational experience for all team members.

Introduction

Beat Fighter is a rhythm-based fighting game. The objective of Beat Fighter is to defeat the orbs, or enemies, flying at you from the top, bottom, left, and right of the game screen by punching them in beat with the music playing. There are a variety of different characters, songs, and backgrounds to choose from the start menu, so the game is customizable every time a song is played. Each song has different choreography, or patterns to follow, in order to beat the level, The choreography of each level is completely unique and varies in difficulty. Beat Fighter was built using Pygame, a library within the coding language Python. Python is a coding language that we learned during the first three and a half days of Operation Catapult, invented in 1989 by Guido van Rossum and commonly used for its readability and the clear logic of its code.

Imports

The first few lines of code within our program are to import the libraries and modules. For Beat Fighter we imported the libraries pygame and time as well as the module sys.

```
1  import pygame
2  import time
3  import sys
```

Classes

```
5   class Dancer:
6       def __init__(self, screen, x, y):...
26
27      def draw(self):...
29
30      def punch_left(self):...
32
33      def punch_right(self):...
35
36      def punch_up(self):...
38
39      def punch_down(self):...
41
42  class Orb:
43      def __init__(self, screen, direction):...
74
75      def draw(self):...
77
78      def hit_by(self, punchDirection):...
80
81      def move(self):...
84
```

A class is a programmer-defined collection of methods and variables which can be called within the main body of the code for various purposes. Within games, it is usually most helpful to use classes to represent an on-screen object, useful to help organize code and to minimize repeated code. There are four major classes within the code for Beat Fighter: Dancer, Orb, HPBar, and Face. Within each class, there are definitions for each essential piece of the class. For the Dancer class, the definitions are __init__ or initialize, draw, punch_left, punch_right, punch_down, and

punch_up. For the Orb class, the definitions are __init__, draw, hit_by, and move. For the HPBar class, the definitions are __init__ and draw. For the Face class, the definitions are __init__ and draw.

```
85
86    class HPBar:
87        def __init__(self, screen):...
91
92        def draw(self):...
97
98
99    class Face:
100       def __init__(self, screen, name):...
109
110       def draw(self):...
```

Defining Main

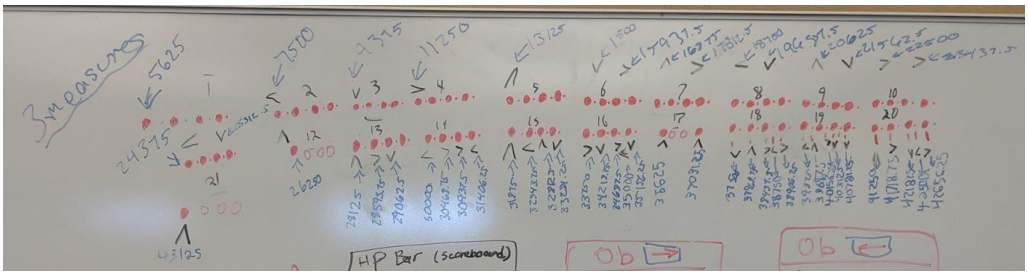
The first thing we do after defining main is to initialize pygame. After this, we start initializing variables. Here, we initialized clock, screen, intro, counselors, songs, song_files, counselor_num, song_num, selection_row, and backflash. “counselors”, “songs”, and “song_files” are lists. The lists for “songs” and “song_files” are tracked using the same integer variable so each song has a unique song file to go with it therefore as long as the song files and songs are in the same

```
133   def main():
134       pygame.init()
135       clock = pygame.time.Clock()
136       # start screen
137       pygame.display.set_caption("Beat Fighter")
138       screen = pygame.display.set_mode((640, 640))
139       intro = True
140       counselors = ["Jared", "Susan", "Jackson"]
141       songs = ["albatraoz.mp3", "old_town_road_diplo.mp3", "EXO Power.mp3", "Chicken Dance.mp3"]
142       song_files = ["albatraoz_bk.txt", "old_town_road.txt", "Power.txt", "chicken_dance.txt"]
143       counselor_num = 0
144       song_num = 0
145       selection_row = 0
146       backflash = False
147       pygame.mixer.music.load("if elevators had trap music.mp3")
148       pygame.mixer.music.play(1, 19)
212           hpbar = HPBar(screen)
213           face = Face(screen, counselors[counselor_num])
214           dancer = Dancer(screen, 90, 90)
215           funished = pygame.image.load("Funished.png")
216           winner = pygame.image.load("victory_screen.png")
217           winner = pygame.transform.scale(winner, (640, 640))
218           pygame.mixer.music.load(songs[song_num])
219           punchbox = (129, 95, 383, 450)
220           hurtbox = (204, 170, 233, 300)
221           orblist = []
222           timeline_dict = {}
230       background_image_frames = []
244       current_image = 0
245       is_game_over = False
246       pygame.mixer.music.play()
247       start_milli_time = int(round(time.time() * 1000))
248       #  main game loop
249       gameplay = True
250       win = False
```

order the song file and the song will be connected. Under the main loop later in the code we also initialized hpbar, face, dancer, funished, winner, punchbox, hurtbox, orblist, timeline_dict, background_image_frames, current_image, is_game_over, start_milli_time, gameplay, and win.

Song Files

The song files are organized by the milliseconds on when the orbs should come to attack the dancer. Originally, only one choreography existed, written on the whiteboard beat by beat for the instrumental of the song “I’m an Albotraoz”. We spent hours figuring out the choreography, the milliseconds on when we wanted the attack to come, and plugging all of this information into the program. We wanted to do more songs, but we were not sure if we would have time. Luckily, one of our amazing teaching assistants, Shijun, helped us make a recorder that would write down the milliseconds and what move we wanted to do whenever we



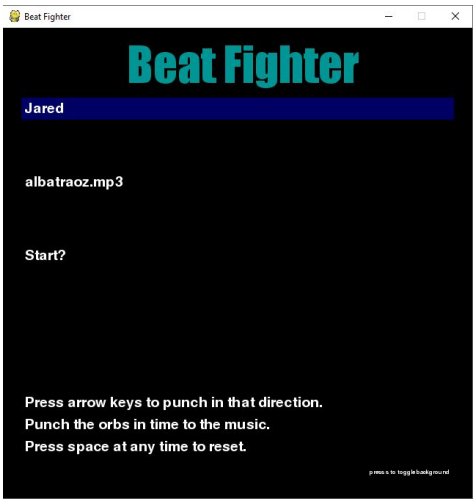
1	5625,down	28	32815,down
2	7500,left	29	33280,up
3	9375,up	30	33750,right
4	11250,right	31	34220,up
5	13125,down	32	34690,right
6	15000,up	33	35000,left
7	15940,right	34	35155,up
8	16875,down	35	35625,down
9	17815,left	36	37030,down
10	18750,left	37	37300,up
11	19690,up	38	37970,down
12	20625,down	39	38440,right
13	21565,up	40	38750,left
14	22500,right	41	38905,right
15	23440,right	42	39375,left
16	24375,left	43	39690,down
17	25315,up	44	40155,left
18	26250,down	45	40315,up
19	28125,down	46	40780,up
20	28595,left	47	41250,down
21	29065,up	48	41720,right
22	30000,left	49	42190,up
23	30470,right	50	42500,left
24	30940,right	51	42655,right
25	31405,left	52	43125,down
26	31875,down	53	44000,over
27	32345,left		

```python
import pygame
import time

def main():

    # initialize the pygame module
    pygame.init()
    screen = pygame.display.set_mode((500, 500))
    f = open('old_town_road.txt', 'w')
    OFFSET = 5
    clock = pygame.time.Clock()
    pygame.mixer.music.load('FRIENDS.mp3')

    pygame.mixer.music.play()
    starting_time = int(round(time.time() * 1000))
    while True:
        clock.tick(250)
        current_time = int(round(time.time() * 1000))
        time_since_start = current_time - starting_time
        rounded_time = time_since_start - (time_since_start % 5) - OFFSET
        print(rounded_time)

        for event in pygame.event.get():
            if event.type == pygame.QUIT:
                pygame.quit()
                return
            if event.type == pygame.KEYDOWN:
                pressed_keys = pygame.key.get_pressed()
                if pressed_keys[pygame.K_UP]:
                    line = str(rounded_time) + ",up\n"
                    f.write(line)
                    print(line)
                elif pressed_keys[pygame.K_DOWN]:
                    line = str(rounded_time) + ",down\n"
                    f.write(line)
                    print(line)
                elif pressed_keys[pygame.K_LEFT]:
                    line = str(rounded_time) + ",left\n"
                    f.write(line)
                    print(line)
                elif pressed_keys[pygame.K_RIGHT]:
                    line = str(rounded_time) + ",right\n"
                    f.write(line)
                    print(line)

        # Clear the screen and set the screen background
        screen.fill((0, 0, 0))

        pygame.display.update()

    f.close()


main()
```

pressed a direction on the arrow keys. This information is written to a .txt file, which is then read and interpreted into the attacks. With the new system, choreographies which took hours before take us fewer than five minutes, and the code which would have composed hundreds of lines now takes up around fifteen.

Start Screen

The start screen is displayed and manipulated within the While Intro loop. This loop draws the rectangle and blits all text after rendering it. By scrolling with the arrow keys, the player may choose their character, song, and background settings before selecting “start”. Pressing up and down selects which list the player will be scrolling through (indicated by the blue rectangle behind the text), while pressing left and right increments through a list until looping back around to the start.




```
251     while gameplay:
252         pressed_keys = pygame.key.get_pressed()
253         if pressed_keys[pygame.K_SPACE]:
254             gameplay = False
255         clock.tick(250)
256         if backflash:...
259         else:
260             screen.fill((0, 0, 0))
261         pygame.draw.rect(screen, (0, 0, 15), punchbox)
262         pygame.draw.rect(screen, (0, 0, 0), hurtbox)
263         for event in pygame.event.get():...
268         hpbar.draw()
269
270         current_milli_time = int(round(time.time() * 1000))
271         time_since_start = current_milli_time - start_milli_time
272         rounded_time = time_since_start - time_since_start % 5
273
274         if rounded_time in timeline_dict:...
285
286         punchway = ''
287         if not is_game_over:
288             pressed_keys = pygame.key.get_pressed()
289             if pressed_keys[pygame.K_DOWN]:...
293             elif pressed_keys[pygame.K_UP]:...
297             elif pressed_keys[pygame.K_LEFT]:...
301             elif pressed_keys[pygame.K_RIGHT]:...
305             else:...
308             face.draw()
309
310             if hpbar == 0:
311                 is_game_over = True
312             # deal with orbs
313             for orb in orblist:...
321             pygame.display.update()
322             for orb in orblist:...
325
326         if hpbar.score <= 0:...
332         if is_game_over:...
```

Game Loop

After the start screen, a few more variables are declared and a few more functions are called before we enter the main game loop. This loop makes use of the classes defined above to achieve its functions. First, it controls the framerate using the time library we imported above, keeping the frames at a stable 250 fps to prevent fluctuations in orb speed. It next draws the selected background, hitboxes, health bar, dancer, face, and orbs. A statement checks if the player is exiting or resetting the game. The statement then checks whether the player is pressing a button, adjusting their punch direction accordingly. An if statement checks if any orbs are in the direction the player is punching and within the hitbox, destroying

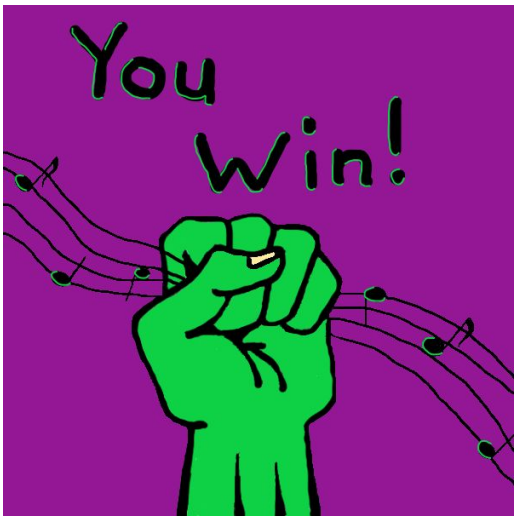
them if they are. The orbs move, and the loop checks if the player is being hit. If they are, the health bar is depleted by 1000. The game checks if the player has won or lost and sets a flag if they are. If they have, a win or lose screen is displayed and music plays. Finally, the screen is updated.

Calling main

Finally, we call the main in a while true loop. This ensures that whenever space is pressed, main will be called again and the game will return to the start screen.

Graphics

Our graphic designer, Cassie, made all of our graphics for the game, including, but not limited to, all five of the character sprites, the victory screen, and our team logo. Firstly, she sketched out the general position and movement each pose or image should have, checking how the shapes and colors would coordinate to create a cohesive picture while still being believable. Secondly, she would then draw each image on her computer, using the MS Paint 3D application and a drawing tablet. From there each image would be uploaded into the program via GitHub and inserted into the game code as necessary. A single character sprite could take several hours of work, from concept art to sketch to color to outlining. The process of creating the background and cover image was very similar. The background animation, a loop of stars flashing by so quickly they're nothing but light trails, is composed of thirteen individual screenshots of a gif, split apart, resized to fit the game screen, and played in rapid succession, to create the animated effect. This is



```
231     background_image_frames.append(pygame.image.load("frame_00.gif"))
232     background_image_frames.append(pygame.image.load("frame_01.gif"))
233     background_image_frames.append(pygame.image.load("frame_02.gif"))
234     background_image_frames.append(pygame.image.load("frame_03.gif"))
235     background_image_frames.append(pygame.image.load("frame_04.gif"))
236     background_image_frames.append(pygame.image.load("frame_05.gif"))
237     background_image_frames.append(pygame.image.load("frame_06.gif"))
238     background_image_frames.append(pygame.image.load("frame_07.gif"))
239     background_image_frames.append(pygame.image.load("frame_08.gif"))
240     background_image_frames.append(pygame.image.load("frame_09.gif"))
241     background_image_frames.append(pygame.image.load("frame_10.gif"))
242     background_image_frames.append(pygame.image.load("frame_11.gif"))
243     background_image_frames.append(pygame.image.load("frame_12.gif"))
```

due to the fact that Python coding does not support gifs, and therefore premade animation will not move, but remain one static frame, if directly uploaded into the program. The cover image, a depiction of the player character punching upward while enemies fly in all different directions, was made in a much simpler way. All the images used to create it were, and because of this uploaded and manipulated without needing anything more than some touch-ups.

Project Reflection

Isabella Patterson:

Our team seemed to work really well together. I can't remember having arguments or problems working together. Allison is really good at coding as a whole so she was able to calmly help many times when we ran into technical difficulties and Cassie and/or I did not understand how to fix the bugs. Cassie is an amazingly talented artist so when we put her on graphics duty we were able to make our game really unique and beautiful. I came in with choreography and rhythm game experience, plus a ton of crazy ideas that my teammates had to reel in. When our team ran into technical difficulties we would ask our teaching assistants, Derek and Shijun, or Dr. Fisher for assistance. It was always understood that since we were so new to Python we were going to need help. Everyone helped not only us create the game we wanted to make, but also explained calmly so we could try to understand what they were telling us. I think my mindset helped the team because I, like my partners, brought creativity and flexibility to the team. None of us went for the leader position, but we all delegated work quite evenly and worked on what we were good at, then what challenged us.

Allison Abernathie:

Our project went very well. The team always got along well and any errors or bugs we had were easily fixed. Even our most challenging, gamebreaking bug took only a few minutes to rectify by restoring a previous version from git. When we first began programming, I was worried that the group might end up relying on me to do all the programming due to my prior experience, but everyone contributed to the code and everyone was always working on something. In addition to overall programming, everyone had preexisting skills they contributed: Cassie used her art experience to work on the graphics, Isabella's dance skills helped her choreograph our dances, and my practice with other languages helped me debug some of the more challenging errors. I went into the project planning to balance helping the team with making my own contributions, and I think that along with the rest of my teammates' attitudes has had a visible positive impact from the overarching idea down to the code itself. I consciously avoided arrogance, a trait I have found to be common in experienced programmers despite being a major hindrance to teamwork and cooperation.

Cassandra Lutes

I am of the opinion that our project went exceedingly well, especially considering the fact that two thirds of the team had little to no former experience coding. Both Isabella and I overcame our inexperience to contribute to the project, and Allison was immensely patient when

we asked for help. We used our resources effectively, asking the faculty for assistance and utilizing the internet when necessary. Each member had a unique skill set that helped contribute something to the project: Allison's knowledge of coding and prior experience made her a tremendous help; Isabella's musical expertise and choreography granted us the ability to make the "dancing" aspect of the game; and my artistic abilities lead to the creation of original art that gives the game a unique style and look. Our collective mindset allowed for the individual pieces to become an effective whole in a timely manner. We came in everyday expecting to work, flexible in our approach but certain that we wanted to see concrete progress. Even with small breaks, this dedication and work ethic meant that not only did we complete the game in time, but we had time to polish it and add extra features. Overall, our project was a wonderful success.