

PvP Fighting Game with Pygame Module

Final Project Report (Algorithm & Programming)



Lyonel Judson Saputra (2802505853)

L1BC

Jude Joseph Lamug Martinez MCS

Binus International
University Jakarta

1. Abstract

This report is the documentation and discussion of a student's Algorithm and Programming class final project in the form of a PvP game. The game is a 2 player fighting game in a 1 vs 1 mode that is made by using the Pygame module. It uses python coding to get player inputs in order to allow them to play against each other in one device, by moving, using attacks, and managing each character's health and knockbacks.

2. Introduction

The purpose of this project is so that students can make their own project by using python coding in order to solve a certain problem. They are to apply what they learn in class and what they learn beyond class to solve the problem. The form of the project can be a game, an app, or a project which pushes students to learn more than what is taught inside of the class. Students are to show their mastery of the python algorithms, programming concepts, and problem solving skills by creating a python solution as a project.

3. Project Inspiration

Creating this PvP game is inspired by a pc/console game called brawlhalla, specifically the 1 v 1 mode of the game. It is a 2 player game that has multiple skills to attack the opponents and have a quite simple movement mechanic. Unlike other PvP games, brawlhalla has the main focus on pushing the opponents off the map in order to win it. Therefore, the skills and attacks in this game are also adjusted to give knockback effects other than just the damage. The movement mechanic also supports multiple jumping and dashing in order to avoid falling off the map. Experiencing these unique mechanics of the game when playing it for the first time on console with my friends makes me want to try and create it with pygame for this project. I thought that instead of making the usual PvP game, I would try and remake this game instead. The fun I had when playing this game and the unique mechanics of the game made me decide to remake this game with pygame for this project.

4. Project Specifications

This game consists of several mechanics such as movement and attack that are meant to be quite close to the game mechanics of the game Brawlhalla. This game gets the user input in order to make the character move and attack according to the player's input. It also sets the health loss of each player and the knockback received based on the attack of the other player. It keeps track of the health of each player by using a health bar. Furthermore, it checks the collision of the players with the map and also applies gravity which makes the characters fall if they do not collide with the ground or platform in the map. This game also provides 3 different attacks which have different amounts of knockback and damage. These game mechanics along with the smooth animation of each character creates an engaging 2D fighting game.

- Framework and modules used for this project:**

- Pygame:** This module is used to get the player inputs, animate the characters, handle the attack logics, and also import the game environment along with the sounds.

- **Python:** The programming language that is used to program all the logic and mechanics of the game
- **This game has several components**
 - Main game loop
 - Character class
 - Game environment
 - HUD
 - User input
- **The Characters in this game have these Attributes:**
 - **Movement**
 - Player 1 (left):
 - Left: A
 - Right: D
 - Jump: W
 - Down: S
 - Dash/Sprint: V
 - Player 2 (right):
 - Left: Left Arrow
 - Right: Right Arrow
 - Jump: Up Arrow
 - Down: Down Arrow
 - Dash/Sprint: M
 - **Attack**
 - Player 1 (left):
 - Attack 1: T
 - Attack 2: G
 - Ultimate: F
 - Player 2 (right):
 - Attack 1: I
 - Attack 2: K
 - Ultimate: L
 - **Character Animation**
 - Each action performed by each character is animated.
 - **HUD**
 - Health bar
 - Sprint bar
 - Skill bar
 - **Game Environment:**

- **Collisions**
 - Ground
 - Platform
- **Camera**
 - Follows the middle of the two players
- **Winning Conditions**
 - The score is 3 or the difference is 2
 - Score is got if
 - Enemy health is 0
 - Goes out of frame either top, bottom, right, left
- **Challenges:**
 - Animation Handling
 - The animation part of the code is the most challenging part of this entire project. Since every single action needs to be animated, it requires a lot of effort to find the right sprite sheets and to cut each of them because there are so many actions that need to be animated. I also needed to edit some of the sprites to fit the actions that are in this project.
- **Future Improvements:**
 - More features
 - Weapons
 - In the actual brawlhalla game, there are weapons that drop randomly and can be used for the player to gain advantage. However, in this project, I was unable to create that feature in time, therefore, it can be something that can be improved on in the future.
 - Character selection
 - There are only 2 characters in this game and it is fixed by default. If this game were to be improved, a character selection feature would be one of the things that can be added to improve the overall game experience.
 - More players (online)
 - Playing with more than 2 players would also be a fun feature to add. However, since it is not possible for 4 players to play on the same pc/laptop, creating an online feature would be a great improvement.
 - More Maps

- There is only one default map in this game, and more maps could be added to add more player customizations to the game which could make them enjoy the game more.

5. Solution Discussion

This game initialises by running the main.py which also imports all the necessary modules to run the game. First of all, the code imports the Pygame module which is used to create the game and also initialises it by using pygame.init() function. It then also imports all the remaining python files of the game like character.py, hud.py, background.py, collision.py which will all be explained more later on. On the main.py, after initializing pygame, the next code sets the screen width and height, and also sets the fps of the game. Then, it loads the background image of the game which is currently the only map for this game. It then starts the main game loop, and sets the key to be pressed to end the loop.

In the game loop, firstly, it calls the update function of the character class which calls all the necessary functions to update each character's attributes throughout the loop. Then, it sets the offset values which will be passed to the background and the characters. Then, it will draw the background that is already modified by the offset and the zoom by using the screen.blit method. Finally, the game loop will call to draw the HUD and the characters itself.

The main game loop also deals with the menu tab which consists of a winning menu and a pause menu. The winning menu will automatically be triggered when one of the players wins which will be explained more in the Character class. The winning menu will be triggered right after the player position is reset which gives it a smooth transition for the players after winning and not abruptly change to the winning menu. In the winning menu, there are 2 buttons that the players can pick, the play again button and the exit button. The play again button resets the game and runs the loop again while the exit button just exits the loop and closes the game. The pause menu will be opened when someone presses the escape button. In the pause menu, there is a resume button which continues the game loop and an exit button which exits the game. These menus are in the menu.py and are imported by the main.py.

The next python file is the collision.py which is simply the file which checks all the collisions of the map. It consists of a Collision class which takes the name of each collision rect in the `__init__` function and creates the rect in the rect function which gets the x position, y position, width, and height for each of the rect. Then, all of the rect is defined by giving it the name, and all the arguments for the rect function. The rects like ground bottom, platform1, etc. will be passed into the Collision class which will create and return all the values of each rect so that it can be used later on to check whether the character is in contact with any of these rects.

Then, there is the background.py file which consists of 3 functions, the zoom function, camera function, and the center_background function. The zoom function gets the background image that is passed from the main game loop where it is called, and returns the background image so that it can be drawn on the screen. The second function is the camera function which deals with the offset of the camera where it should be in the middle of the two characters when they are moving. It gets the initial midpoint coordinates of the two players and subtracts it with the live average of the two x and y positions which gives the live coordinates of the midpoint of the two characters. This mechanism causes the camera to be able to stay in the middle of these two players and move dynamically with them. It also returns the x and y offset to the main game loop to be passed on to the other elements of the game. The third function is the center_background function which gets the zoomed image and produces the centered zoomed image. This centered image will be then returned to the main game loop.

Now that the background and the collision is set up, let's discuss the longest and most important part of the entire code, the character.py file which contains the Character class. This file imports the collision.py file because it requires the checking of the collision between the player and the map. The first few lines of code just initializes the screen, and also the pygame mixer which will be used to generate the sound for the game. Then, this file starts to load all the needed elements like the audio, but also all the animation frames that are required to animate the actions of each character. There are 12 actions for each character that needs to be imported and each action has multiple frames varying from 4 frames to 14 frames. After all the animation frames are loaded, the Character class is defined and takes 5 arguments from the character1 and character2 variables that are to be defined later. In the `__init__`, it has a lot of instance variables which takes care of a lot of things the characters do like the movement, collision checking, attacking, animation, etc.

The first function of the Character class is a get function which is the `get_rect_hitbox` which returns the hitbox rect of the character. It along with the second function, which returns the range rect, will be used to detect collisions and detect the range of the character's attacks. After that, there is the `take_damage` function which subtracts the current character health by the damage it took. It also calls the `check_health` function which is the next function in the class. The `check_health` function just checks whether the character has health or not and decides what to do in either situation. If the character's health is 0, then it will start the reset process, and set the animation logic to show the death animation along with playing the death sound effect. On the contrary, when the character's health is above 0, then it will have the original character attributes and the player can continue playing like normal.

After that, there is the `check_collision` function which checks whether the character's hitbox rect is colliding with the map such as ground, platforms, and

borders which is obtained from the collision file discussed before. First of all, this function checks the borders which are the top, bottom, left, and right borders of the screen, and if the character makes contact with it, their health will be set to 0. Since there are 4 borders, it uses the for loop which checks all three of them and if either of the three collides with the character hitbox rect. If any of them collides, then the character will automatically be reset and the other character will get a point. Then, it checks the ground and platforms which would stop the character from falling if colliding with it since there is a gravity mechanism in this game. Upon colliding with the ground or the platforms, the character will stop falling.

The next function is the check_movement function whose purpose is to decide which action's animation is prioritised, for example the attack animation has to override the walking animation for it to make sense when playing the game. All of them use initial variables that are triggered by each input or action the character performs and call another function that sets the current_action of the character which is directly linked to the animation. It also checks for the direction of the character to set the animation for the attack logic.

The movement function handles the user input for moving the characters. It has two controls, the left one, and the right one. The left one consists of wasd to move up, down, right, and left while the right one uses the arrow keys. The jumping mechanism makes use of the gravity mechanics in this game and adds the character's y position which makes it jump and the gravity mechanic pulls it back down. The left and right movement is just adding to the character's x position. The dash logic makes the jump and walk function much faster, and there is a limited time to use it. The movement function also calls the check_movement function that is mentioned before in order to set the animation of each movement. Lastly, it also calls on the collision function that is also discussed before to check the collision of the character and the collision elements.

The reset position function resets the character's position and health after they fall off the map or their health is 0. This function does not teleport the character to the original position, however, it gradually transports them by dividing the x and y distance from the character's current position and the target position by 30, and then incrementing the x and y position of the character by the value that is obtained. Then, it will stop once it gets close to the respawn position and reset all the player attributes so that the player can continue to play.

The next one is the check_hits function which checks whether a character is in range of the other and it does it by using the colliderect between the character's hitbox and the opponent's range rect. It then returns whether or not the character is in range. The next function is the player_direction function which just gets the player direction according to the movement function that was discussed before and uses the player direction to modify the size and position of the range rect. It modifies it for two

situations, which are melee and range, and in both of these situations, the range rect will be modified differently.

After that, there is the knockback function which sets the knockback amount and knockback direction when a character is hit by the other. It only sets the knockback direction if there is no previous knockback direction in order to prevent the knockback from changing directions. Then, it also sets the multipliers according to the attack that the player uses, and also the health loss of the character. If the health lost is more, then the knockback amount will also be more. After that, it checks the knockback direction that is set and changes the x or y position of the character accordingly. Then, it decelerates the knockback by dividing the knockback amount by a certain amount continuously until it stops. Once the knockback stops, all the movement and attack character attributes are returned to normal.

The attack function handles the attack based on the input of the player. It then sets the damage that will be dealt to the other character, and also the knockback amount after checking if the enemy is in range and if the counter is full already. The first attack is a basic attack that can be used whenever, but has low knockback and damage, and is also melee. The second attack has a higher damage and knockback, and is a ranged attack, which requires 3 times hit with the basic attack. This is done by setting the range to ‘range’ so that the range rect will be edited to support a ranged attack. The ultimate does the most knockback, but requires 8 times hit with the basic attack and is a melee attack. The attack function also sets the instance variables that are used in the check_movement function according to the attack that the character does. It then animates the current attack. It also plays the sound effect when pressing any attack.

The next function is the score function which adds the score based on the instance variable that is set by the other functions and determines the winning condition for each character. It just uses 2 if statements and some ‘and’ and ‘or’. The set_action statement sets the action of the character which will be used for the animation.

The update_animation function determines the current frame index from the animation frames to draw by the draw function and adds the frame index by a certain speed which determines how fast the animation is or how fast the frames change. It is the code that handles the logic for the animation of the character’s actions. The speed of the animation change for each character’s actions is also set within this function. It also sets certain actions like walk and idle on a loop by setting the frame index to 0 again after reaching the final frame index of that action. This function also handles the flipping of the image when the direction is left and right.

The draw function gets the current frame index from the update_animation function and draws the image corresponding to it. This function also further modify the offsets so that the animation images will fit to the character rects so that the

gameplay will be better and to make sure that the animation is in sync with the gameplay. Finally, there is the reset function which is called when one of the character's wins and they want to play again, and also the update function which calls every necessary function from the Character class. Overall, the Character class handles everything the character does and keeps track of all the character attributes.

The character1 and character2 is then defined to pass all the arguments that are inside of the Character class. It also uses a dictionary to pass the animation argument for the Character class. It also helps to decide which animation is referred to in the update animation and set action function.

The hud.py file handles all the HUD elements of the game such as the health bar, skill bar, etc. These are linked to the Character class because the HUD elements get their values from the character attributes. The hud.py file contains the draw_hud function which is responsible for getting the character attributes and setting each hud element like the health bar. The first few lines on this function sets the fonts and the bar colors for each hud element for the first character. It also sets the size for the health bar like the height and width of it. Then, it sets the max arc, thickness, and radius for the skill arc. After everything is set, the function then draws the rect of the health bar by using all the set items and also giving the x and y position of the health bar. The health bar itself has 2 bars, one is the background bar which is slightly bigger and has a constant size, while the other one is the bar that shows the actual amount of health the character has. The health bar also writes the number of the character's current health.

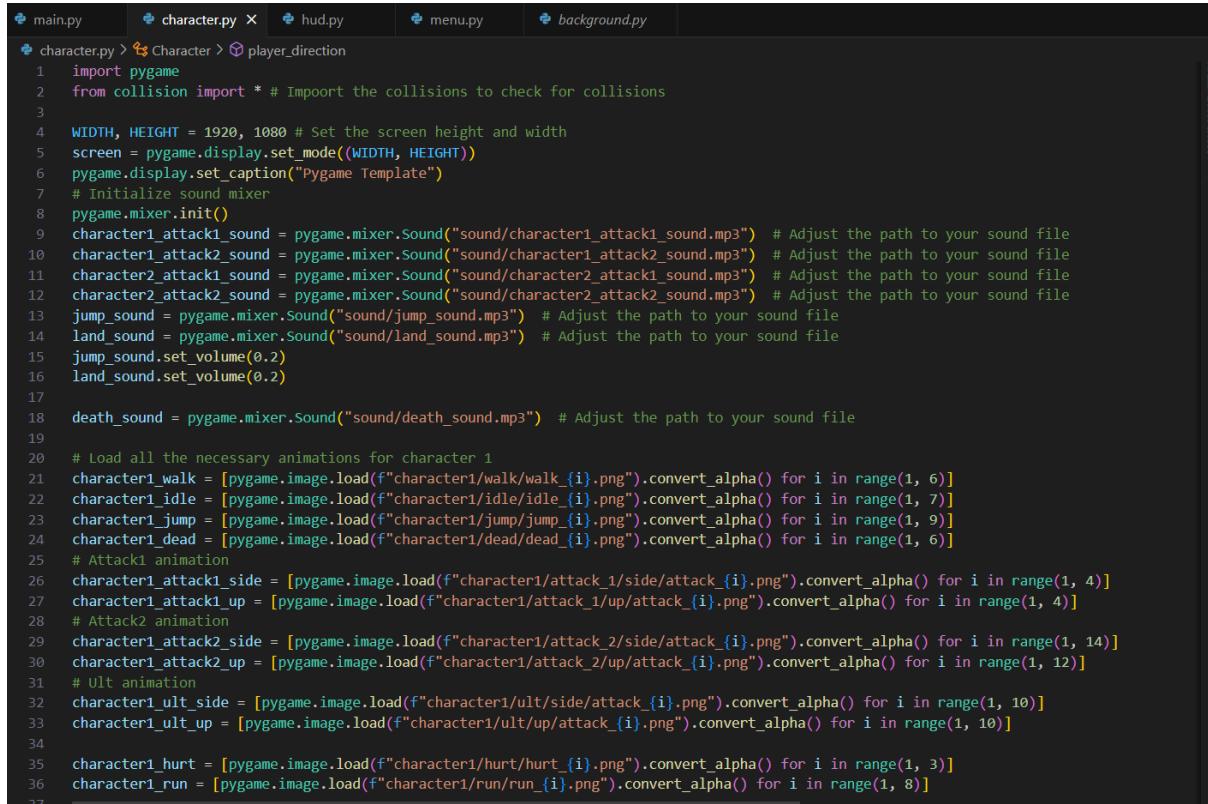
Besides the health bar, the draw_hud function also draws the arc for the second attack and the ultimate. It also has 2 circles, one for the background and one for the actual skill bar. The main skill arc divides the attack 2 counter by 3 because it takes 3 hits to be full and divides the ultimate counter by 8 because it takes 8 hits to be full. So, it fills $\frac{1}{3}$ of the arc for the second attack and fills $\frac{1}{8}$ of the arc for the ultimate. These skill arcs show when the special attacks are ready. The HUD function also shows the score of each player which it gets the data from the Character class. It also shows the sprint bar which tells the player how much sprint they have left. The draw_hud function also removes all the mentioned HUD items and shows a winning text when one of the players wins, which goes hand in hand with the code I discussed in the main game loop.

6. Evidence

Image 1: main.py: Importing Pygame and running the main loop

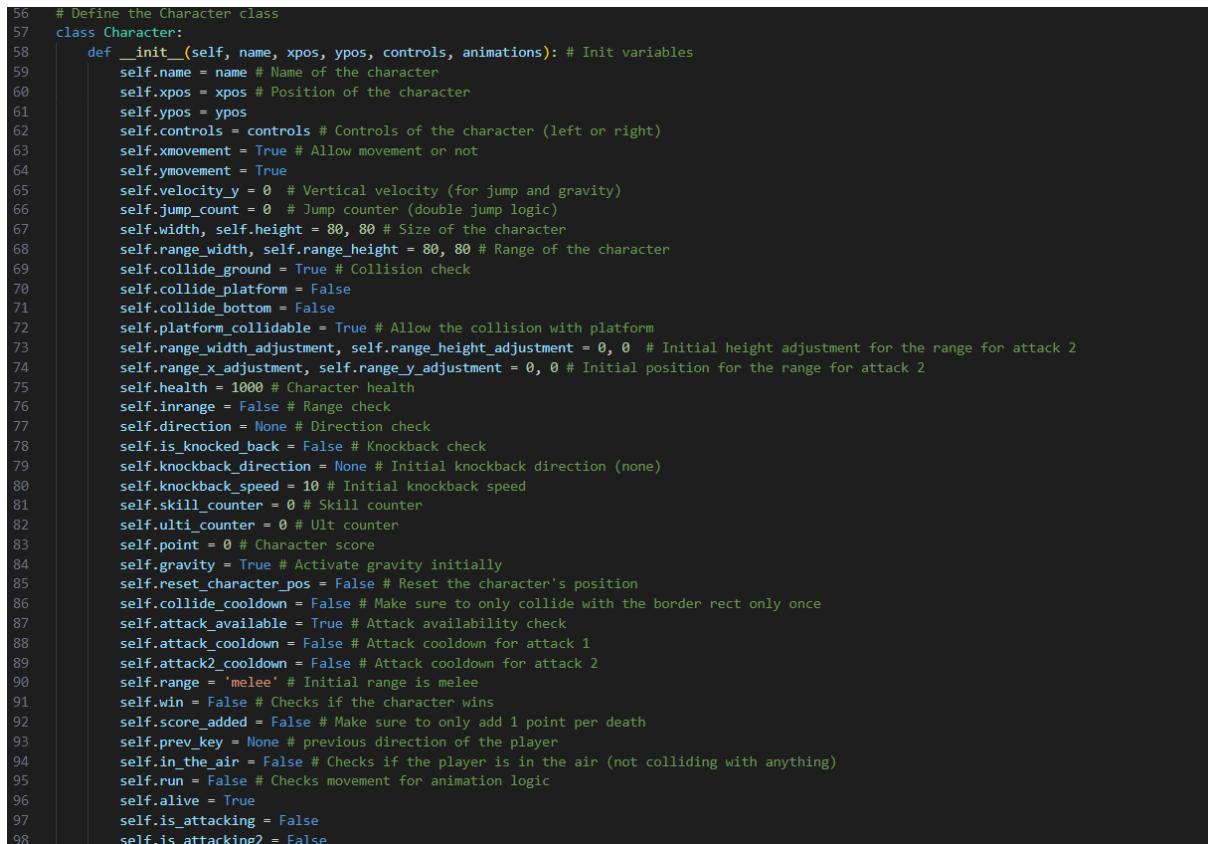
```
main.py > ...
1 import pygame
2 import sys
3 from character import *
4 from hud import *
5 from background import *
6 from collision import *
7 from menu import show_menu, pause_menu
8
9 # Initialize Pygame
10 pygame.init()
11
12 # Screen dimensions
13 WIDTH, HEIGHT = 1920, 1080
14 screen = pygame.display.set_mode((WIDTH, HEIGHT))
15 pygame.display.set_caption("Pygame Template")
16
17 # Frame rate
18 FPS = 120
19 clock = pygame.time.Clock()
20
21 # Load background image
22 background_image = pygame.image.load("map.webp").convert_alpha()
23
24 # Main game loop
25 if __name__ == '__main__':
26     running = True
27     while running:
28         # Reset game
29         character1.reset()
30         character2.reset()
31
32         while True: # Inner game loop
33             for event in pygame.event.get():
34                 if event.type == pygame.QUIT:
35                     running = False
36                     break
37                 elif event.type == pygame.KEYDOWN:
```

Image 2: character.py: Importing and Loading all the animation and audio



```
main.py character.py x hud.py menu.py background.py
character.py > Character > player_direction
1 import pygame
2 from collision import * # Import the collisions to check for collisions
3
4 WIDTH, HEIGHT = 1920, 1080 # Set the screen height and width
5 screen = pygame.display.set_mode((WIDTH, HEIGHT))
6 pygame.display.set_caption("Pygame Template")
7 # Initialize sound mixer
8 pygame.mixer.init()
9 character1_attack1_sound = pygame.mixer.Sound("sound/character1_attack1_sound.mp3") # Adjust the path to your sound file
10 character1_attack2_sound = pygame.mixer.Sound("sound/character1_attack2_sound.mp3") # Adjust the path to your sound file
11 character2_attack1_sound = pygame.mixer.Sound("sound/character2_attack1_sound.mp3") # Adjust the path to your sound file
12 character2_attack2_sound = pygame.mixer.Sound("sound/character2_attack2_sound.mp3") # Adjust the path to your sound file
13 jump_sound = pygame.mixer.Sound("sound/jump_sound.mp3") # Adjust the path to your sound file
14 land_sound = pygame.mixer.Sound("sound/land_sound.mp3") # Adjust the path to your sound file
15 jump_sound.set_volume(0.2)
16 land_sound.set_volume(0.2)
17
18 death_sound = pygame.mixer.Sound("sound/death_sound.mp3") # Adjust the path to your sound file
19
20 # Load all the necessary animations for character 1
21 character1_walk = [pygame.image.load(f"character1/walk/walk_{i}.png").convert_alpha() for i in range(1, 6)]
22 character1_idle = [pygame.image.load(f"character1/idle/idle_{i}.png").convert_alpha() for i in range(1, 7)]
23 character1_jump = [pygame.image.load(f"character1/jump/jump_{i}.png").convert_alpha() for i in range(1, 9)]
24 character1_dead = [pygame.image.load(f"character1/dead/dead_{i}.png").convert_alpha() for i in range(1, 6)]
25 # Attack1 animation
26 character1_attack1_side = [pygame.image.load(f"character1/attack_1/side/attack_{i}.png").convert_alpha() for i in range(1, 4)]
27 character1_attack1_up = [pygame.image.load(f"character1/attack_1/up/attack_{i}.png").convert_alpha() for i in range(1, 4)]
28 # Attack2 animation
29 character1_attack2_side = [pygame.image.load(f"character1/attack_2/side/attack_{i}.png").convert_alpha() for i in range(1, 14)]
30 character1_attack2_up = [pygame.image.load(f"character1/attack_2/up/attack_{i}.png").convert_alpha() for i in range(1, 12)]
31 # Ult animation
32 character1_ult_side = [pygame.image.load(f"character1/ult/side/attack_{i}.png").convert_alpha() for i in range(1, 10)]
33 character1_ult_up = [pygame.image.load(f"character1/ult/up/attack_{i}.png").convert_alpha() for i in range(1, 10)]
34
35 character1_hurt = [pygame.image.load(f"character1/hurt/hurt_{i}.png").convert_alpha() for i in range(1, 3)]
36 character1_run = [pygame.image.load(f"character1/run/run_{i}.png").convert_alpha() for i in range(1, 8)]
```

Image 3: character.py: Character class



```
56 # Define the Character class
57 class Character:
58     def __init__(self, name, xpos, ypos, controls, animations): # Init variables
59         self.name = name # Name of the character
60         self.xpos = xpos # Position of the character
61         self.ypos = ypos
62         self.controls = controls # Controls of the character (left or right)
63         self.xmovement = True # Allow movement or not
64         self.ymovement = True
65         self.velocity_y = 0 # Vertical velocity (for jump and gravity)
66         self.jump_count = 0 # Jump counter (double jump logic)
67         self.width, self.height = 80, 80 # Size of the character
68         self.range_width, self.range_height = 80, 80 # Range of the character
69         self.collide_ground = True # Collision check
70         self.collide_platform = False
71         self.collide_bottom = False
72         self.platform_collidable = True # Allow the collision with platform
73         self.range_width_adjustment, self.range_height_adjustment = 0, 0 # Initial height adjustment for the range for attack 2
74         self.range_x_adjustment, self.range_y_adjustment = 0, 0 # Initial position for the range for attack 2
75         self.health = 1000 # Character health
76         self.inrange = False # Range check
77         self.direction = None # Direction check
78         self.is_knocked_back = False # Knockback check
79         self.knockback_direction = None # Initial knockback direction (none)
80         self.knockback_speed = 10 # Initial knockback speed
81         self.skill_counter = 0 # Skill counter
82         self.ulti_counter = 0 # Ult counter
83         self.point = 0 # Character score
84         self.gravity = True # Activate gravity initially
85         self.reset_character_pos = False # Reset the character's position
86         self.collide_cooldown = False # Make sure to only collide with the border rect only once
87         self.attack_available = True # Attack availability check
88         self.attack_cooldown = False # Attack cooldown for attack 1
89         self.attack2_cooldown = False # Attack cooldown for attack 2
90         self.range = 'melee' # Initial range is melee
91         self.win = False # Checks if the character wins
92         self.score_added = False # Make sure to only add 1 point per death
93         self.prev_key = None # previous direction of the player
94         self.in_the_air = False # Checks if the player is in the air (not colliding with anything)
95         self.run = False # Checks movement for animation logic
96         self.alive = True
97         self.is_attacking = False
98         self.is_attacking2 = False
```

Image 5: character.py: Character rect and health

```

0 # Create the character's hitbox rect
1 def get_rect_hitbox(self):
2     return pygame.Rect(self.xpos + 30, self.ypos , self.width - 40, self.height)
3
4 # Create the character's range rect
5 def get_rect_range(self):
6     return pygame.Rect(self.xpos + self.range_x_adjustment + 30, self.ypos - self.range_y_adjustment, self.range_width - self.range_width_adjustment - 40, self.range_height - self.range_height_adjustment)
7
8 # Sets the health of the character based on the damage taken
9 def take_damage(self, damage):
10     self.health = max(0, self.health - damage) # Health can't go below 0
11
12     self.check_health()
13
14 # Checks the health of the character
15 def check_health(self):
16     if self.health == 0: # Checks if the health is 0
17         self.alive = False # Adjusts the character attributes when losing a round
18         self.set_action('dead')
19         self.reset_character_pos = True
20         selfulti_counter = 0
21         self.movement = False
22         self.ymovement = False
23         character1.attack_available, character2.attack_available = False, False
24         self.in_the_air = False
25         death_sound.play()
26
27     elif self.health > 0: # Restore the character attributes when the health is more than 0
28         self.alive = True
29         self.set_action('idle')
30         self.reset_character_pos = False
31         self.movement = True
32         self.ymovement = True
33         character1.attack_available, character2.attack_available = True, True
34
35 # Checks the character hitbox rect collision with the map
36 def check_collision(self, ground, platforms, borders):
37     for border in borders:
38         if self.get_rect_hitbox().colliderect(border) and self.collide_cooldown == False:
39             self.reset_character_pos = True
40             self.collide_cooldown = True
41             self.health = 0
42             self.check_health()
43
44     if self.get_rect_hitbox().colliderect(ground):
45         self.ypos = ground.top - self.height
46         self.velocity_y = 0
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
649
650
651
652
653
654
655
656
657
658
659
659
660
661
662
663
664
665
666
667
668
669
669
670
671
672
673
674
675
676
677
678
679
679
680
681
682
683
684
685
686
687
687
688
689
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
709
710
711
712
713
714
715
716
717
718
719
719
720
721
722
723
724
725
726
727
728
729
729
730
731
732
733
734
735
736
737
738
739
739
740
741
742
743
744
745
746
747
747
748
749
749
750
751
752
753
754
755
756
757
758
759
759
760
761
762
763
764
765
766
767
767
768
769
769
770
771
772
773
774
775
776
777
777
778
779
779
780
781
782
783
784
785
785
786
787
787
788
789
789
790
791
792
793
794
795
795
796
797
797
798
799
799
800
801
802
803
803
804
805
805
806
807
807
808
809
809
810
811
811
812
812
813
813
814
814
815
815
816
816
817
817
818
818
819
819
820
820
821
821
822
822
823
823
824
824
825
825
826
826
827
827
828
828
829
829
830
830
831
831
832
832
833
833
834
834
835
835
836
836
837
837
838
838
839
839
840
840
841
841
842
842
843
843
844
844
845
845
846
846
847
847
848
848
849
849
850
850
851
851
852
852
853
853
854
854
855
855
856
856
857
857
858
858
859
859
860
860
861
861
862
862
863
863
864
864
865
865
866
866
867
867
868
868
869
869
870
870
871
871
872
872
873
873
874
874
875
875
876
876
877
877
878
878
879
879
880
880
881
881
882
882
883
883
884
884
885
885
886
886
887
887
888
888
889
889
890
890
891
891
892
892
893
893
894
894
895
895
896
896
897
897
898
898
899
899
900
900
901
901
902
902
903
903
904
904
905
905
906
906
907
907
908
908
909
909
910
910
911
911
912
912
913
913
914
914
915
915
916
916
917
917
918
918
919
919
920
920
921
921
922
922
923
923
924
924
925
925
926
926
927
927
928
928
929
929
930
930
931
931
932
932
933
933
934
934
935
935
936
936
937
937
938
938
939
939
940
940
941
941
942
942
943
943
944
944
945
945
946
946
947
947
948
948
949
949
950
950
951
951
952
952
953
953
954
954
955
955
956
956
957
957
958
958
959
959
960
960
961
961
962
962
963
963
964
964
965
965
966
966
967
967
968
968
969
969
970
970
971
971
972
972
973
973
974
974
975
975
976
976
977
977
978
978
979
979
980
980
981
981
982
982
983
983
984
984
985
985
986
986
987
987
988
988
989
989
990
990
991
991
992
992
993
993
994
994
995
995
996
996
997
997
998
998
999
999
1000
1000
1001
1001
1002
1002
1003
1003
1004
1004
1005
1005
1006
1006
1007
1007
1008
1008
1009
1009
1010
1010
1011
1011
1012
1012
1013
1013
1014
1014
1015
1015
1016
1016
1017
1017
1018
1018
1019
1019
1020
1020
1021
1021
1022
1022
1023
1023
1024
1024
1025
1025
1026
1026
1027
1027
1028
1028
1029
1029
1030
1030
1031
1031
1032
1032
1033
1033
1034
1034
1035
1035
1036
1036
1037
1037
1038
1038
1039
1039
1040
1040
1041
1041
1042
1042
1043
1043
1044
1044
1045
1045
1046
1046
1047
1047
1048
1048
1049
1049
1050
1050
1051
1051
1052
1052
1053
1053
1054
1054
1055
1055
1056
1056
1057
1057
1058
1058
1059
1059
1060
1060
1061
1061
1062
1062
1063
1063
1064
1064
1065
1065
1066
1066
1067
1067
1068
1068
1069
1069
1070
1070
1071
1071
1072
1072
1073
1073
1074
1074
1075
1075
1076
1076
1077
1077
1078
1078
1079
1079
1080
1080
1081
1081
1082
1082
1083
1083
1084
1084
1085
1085
1086
1086
1087
1087
1088
1088
1089
1089
1090
1090
1091
1091
1092
1092
1093
1093
1094
1094
1095
1095
1096
1096
1097
1097
1098
1098
1099
1099
1100
1100
1101
1101
1102
1102
1103
1103
1104
1104
1105
1105
1106
1106
1107
1107
1108
1108
1109
1109
1110
1110
1111
1111
1112
1112
1113
1113
1114
1114
1115
1115
1116
1116
1117
1117
1118
1118
1119
1119
1120
1120
1121
1121
1122
1122
1123
1123
1124
1124
1125
1125
1126
1126
1127
1127
1128
1128
1129
1129
1130
1130
1131
1131
1132
1132
1133
1133
1134
1134
1135
1135
1136
1136
1137
1137
1138
1138
1139
1139
1140
1140
1141
1141
1142
1142
1143
1143
1144
1144
1145
1145
1146
1146
1147
1147
1148
1148
1149
1149
1150
1150
1151
1151
1152
1152
1153
1153
1154
1154
1155
1155
1156
1156
1157
1157
1158
1158
1159
1159
1160
1160
1161
1161
1162
1162
1163
1163
1164
1164
1165
1165
1166
1166
1167
1167
1168
1168
1169
1169
1170
1170
1171
1171
1172
1172
1173
1173
1174
1174
1175
1175
1176
1176
1177
1177
1178
1178
1179
1179
1180
1180
1181
1181
1182
1182
1183
1183
1184
1184
1185
1185
1186
1186
1187
1187
1188
1188
1189
1189
1190
1190
1191
1191
1192
1192
1193
1193
1194
1194
1195
1195
1196
1196
1197
1197
1198
1198
1199
1199
1200
1200
1201
1201
1202
1202
1203
1203
1204
1204
1205
1205
1206
1206
1207
1207
1208
1208
1209
1209
1210
1210
1211
1211
1212
1212
1213
1213
1214
1214
1215
1215
1216
1216
1217
1217
1218
1218
1219
1219
1220
1220
1221
1221
1222
1222
1223
1223
1224
1224
1225
1225
1226
1226
1227
1227
1228
1228
1229
1229
1230
1230
1231
1231
1232
1232
1233
1233
1234
1234
1235
1235
1236
1236
1237
1237
1238
1238
1239
1239
1240
1240
1241
1241
1242
1242
1243
1243
1244
1244
1245
1245
1246
1246
1247
1247
1248
1248
1249
1249
1250
1250
1251
1251
1252
1252
1253
1253
1254
1254
1255
1255
1256
1256
1257
1257
1258
1258
1259
1259
1260
1260
1261
1261
1262
1262
1263
1263
1264
1264
1265
1265
1266
1266
1267
1267
1268
1268
1269
1269
1270
1270
1271
1271
1272
1272
1273
1273
1274
1274
1275
1275
1276
1276
1277
1277
1278
1278
1279
1279
1280
1280
1281
1281
1282
1282
1283
1283
1284
1284
1285
1285
1286
1286
1287
1287
1288
1288
1289
1289
1290
1290
1291
1291
1292
1292
1293
1293
1294
1294
1295
1295
1296
1296
1297
1297
1298
1298
1299
1299
1300
1300
1301
1301
1302
1302
1303
1303
1304
1304
1305
1305
1306
1306
1307
1307
1308
1308
1309
1309
1310
1310
1311
1311
1312
1312
1313
1313
1314
1314
1315
1315
1316
1316
1317
1317
1318
1318
1319
1319
1320
1320
1321
1321
1322
1322
1323
1323
1324
1324
1325
1325
1326
1326
1327
1327
1328
1328
1329
1329
1330
1330
1331
1331
1332
1332
1333
1333
1334
1334
1335
1335
1336
1336
1337
1337
1338
1338
1339
1339
1340
1340
1341
1341
1342
1342
1343
1343
1344
1344
1345
1345
1346
1346
1347
1347
1348
1348
1349
1349
1350
1350
1351
1351
1352
1352
1353
1353
1354
1354
1355
1355
1356
1356
1357
1357
1358
1358
1359
1359
1360
1360
1361
1361
1362
1362
1363
1363
1364
1364
1365
1365
1366
1366
1367
1367
1368
1368
1369
1369
1370
1370
1371
1371
1372
1372
1373
1373
1374
1374
1375
1375
1376
1376
1377
1377
1378
1378
1379
1379
1380
1380
1381
1381
1382
1382
1383
1383
1384
1384
1385
1385
1386
1386
1387
1387
1388
1388
1389
1389
1390
1390
1391
1391
1392
1392
1393
1393
1394
1394
1395
1395
1396
1396
1397
1397
1398
1398
1399
1399
1400
1400
1401
1401
1402
1402
1403
1403
1404
1404
1405
1405
1406
1406
1407
1407
1408
1408
1409
1409
1410
1410
1411
1411
1412
1412
1413
1413
1414
1414
1415
1415
1416
1416
1417
1417
1418
1418
1419
1419
1420
1420
1421
1421
1422
1422
1423
1423
1424
1424
1425
1425
1426
1426
1427
1427
1428
1428
1429
1429
1430
1430
1431
1431
1432
1432
1433
1433
1434
1434
1435
1435
1436
1436
1437
1437
1438
1438
1439
1439
1440
1440
1441
1441
1442
1442
1443
1443
1444
1444
1445
1445
1446
1446
1447
1447
1448
1448
1449
1449
1450
1450
1451
1451
1452
1452
1453
1453
1454
1454
1455
1455
1456
1456
1457
1457
1458
1458
1459
1459
1460
1460
1461
1461
1462
1462
1463
1463
1464
1464
1465
1465
1466
1466
1467
1467
1468
1468
1469
1469
1470
1470
1471
1471
1472
1472
1473
1473
1474
1474
1475
1475
1476
1476
1477
1477
1478
1478
1479
1479
1480
1480
1481
1481
1482
1482
1483
1483
1484
1484
1485
1485
1486
1486
1487
1487
1488
1488
1489
1489
1490
1490
1491
1491
1492
1492
1493
1493
1494
1494
1495
1495
1496
1496
1497
1497
1498
1498
1499
1499
1500
1500
1501
1501
1502
1502
1503
1503
1504
1504
1505
1505
1506
1506
1507
1507
1508
1508
1509
1509
1510
1510
1511
1511
1512
1512
1513
1513
1514
1514
1515
1515
1516
1516
1517
1517
1518
1518
1519
1519
1520
1520
1521
1521
1522
1522
1523
1523
1524
1524
1525
1525
1526
1526
1527
1527
1528
1528
1529
1529
1530
1530
1531
1531
1532
1532
1533
1533
1534
1534
1535
1535
1536
1536
1537
1537
1538
1538
1539
1539
1540
1540
1541
1541
1542
1542
1543
1543
1544
1544
1545
1545
1546
1546
1547
1547
1548
1548
1549
1549
1550
1550
1551
1551
1552
1552
1553
1553
1554
1554
1555
1555
1556
1556
1557
1557
1558
1558
1559
1559
1560
1560
1561
1561
1562
1562
1563
1563
1564
1564
1565
1565
1566
1566
1567
1567
1568
1568
1569
1569
1570
1570
1571
1571
1572
1572
1573
1573
1574
1574
1575
1575
1576
1576
1577
1577
1578
1578
1579
1579
1580
1580
1581
1581
1582
1582
1583
1583
1584
1584
1585
1585
1586
1586
1587
1587
1588
1588
1589
1589
1590
1590
1591
1591
1592
1592
1593
1593
1594
1594
1595
1595
1596
1596
1597
1597

```

Image 8: character.py: Character movement

```
# Handles character movement
def movement(self):
    keys = pygame.key.get_pressed() # Get inputs from the player

    # Left character controls
    if self.controls == 'left':
        if keys[pygame.K_w] and self.velocity_y >= 0 and self.ymovement: # Character 1 jump
            self.direction = 'up'
            if self.jump_count < 2:
                if not self.hurt:
                    self.velocity_y = -16
                if self.hurt:
                    self.velocity_y = -10
                self.jump_count += 1
                self.in_the_air = True
                self.frame_index = 0
                self.platform_collidable = True
                jump_sound.play()
                self.sound_played = False
            if keys[pygame.K_v] and self.velocity_y >= 18 and self.sprint_time > 0: # Dash
                self.velocity_y = -20
                self.sprint_time -= 200
                self.run = True
                jump_sound.play()

        if keys[pygame.K_s] and self.ymovement: # Character 1 move down
            self.direction = 'down'
            if self.collide_platform:
                self.in_the_air = True
                self.frame_index = 5
                self.velocity_y += 0.78
                self.ypos += self.velocity_y
                self.platform_collidable = False
                self.sound_played = False

            if keys[pygame.K_v] and self.sprint_time > 0: # Dash
                self.velocity_y = 25
                self.sprint_time -= 200
                self.run = True
                jump_sound.play()

        if keys[pygame.K_a] and not self.collide_bottom and self.ymovement: # Character 1 move left
```

Image 9: character.py: Character reset position

```
# Reset the character's position if their health is 0 or fall off the map
def reset_position(self, target_x, target_y):
    self.gravity = False # Stop gravity to reset the character's position
    self.velocity_y = 0 # Reset vertical velocity
    self.beam_height = HEIGHT
    self.beam_width = 120
    self.transparency = 50
        # Calculate the distance to the target
    x_step = abs(self.xpos - target_x) / 30
    y_step = abs(self.ypos - target_y) / 30

        # Move horizontally towards target_x
    if self.xpos < target_x:
        self.xpos += x_step
    elif self.xpos > target_x:
        self.xpos -= x_step

        # Move vertically towards target_y
    if self.ypos < target_y:
        self.ypos += y_step
    elif self.ypos > target_y:
        self.ypos -= y_step

        # When reaching the target, restore initial character attributes
    if abs(self.xpos - target_x) < 5 and abs(self.ypos - target_y) < 5:
        self.xpos, self.ypos = target_x, target_y
        self.gravity = True # Reactivate gravity
        self.reset_character_pos = False
        self.collide_cooldown = False
        self.health = 1000
        self.check_health()
        self.score_added = False
```

Image 10: character.py: Character check hits and direction

```

# Reset the character's position if their health is 0 or fall off the map
def reset_position(self, target_x, target_y):
    self.gravity = False # Stop gravity to reset the character's position
    self.velocity_y = 0 # Reset vertical velocity
    self.beam_height = HEIGHT
    self.beam_width = 120
    self.transparency = 50
        # Calculate the distance to the target
    x_step = abs(self.xpos - target_x) / 30
    y_step = abs(self.ypos - target_y) / 30

        # Move horizontally towards target_x
    if self.xpos < target_x:
        self.xpos += x_step
    elif self.xpos > target_x:
        self.xpos -= x_step

        # Move vertically towards target_y
    if self.ypos < target_y:
        self.ypos += y_step
    elif self.ypos > target_y:
        self.ypos -= y_step

        # When reaching the target, restore initial character attributes
    if abs(self.xpos - target_x) < 5 and abs(self.ypos - target_y) < 5:
        self.xpos, self.ypos = target_x, target_y
        self.gravity = True # Reactivate gravity
        self.reset_character_pos = False
        self.collide_cooldown = False
        self.health = 1000
        self.check_health()
        self.score_added = False

```

Image 11: character.py: Character knockback

```

# Reset the character's position if their health is 0 or fall off the map
def reset_position(self, target_x, target_y):
    self.gravity = False # Stop gravity to reset the character's position
    self.velocity_y = 0 # Reset vertical velocity
    self.beam_height = HEIGHT
    self.beam_width = 120
    self.transparency = 50
        # Calculate the distance to the target
    x_step = abs(self.xpos - target_x) / 30
    y_step = abs(self.ypos - target_y) / 30

        # Move horizontally towards target_x
    if self.xpos < target_x:
        self.xpos += x_step
    elif self.xpos > target_x:
        self.xpos -= x_step

        # Move vertically towards target_y
    if self.ypos < target_y:
        self.ypos += y_step
    elif self.ypos > target_y:
        self.ypos -= y_step

        # When reaching the target, restore initial character attributes
    if abs(self.xpos - target_x) < 5 and abs(self.ypos - target_y) < 5:
        self.xpos, self.ypos = target_x, target_y
        self.gravity = True # Reactivate gravity
        self.reset_character_pos = False
        self.collide_cooldown = False
        self.health = 1000
        self.check_health()
        self.score_added = False

```

Image 12: character.py: Character attack

```

# Handles the character's attack
def attack(self, other):

    keys = pygame.key.get_pressed() # Get the user input
    multiplier = 1 # multiplier of knockback
    # Left character attack controls
    if self.controls == 'left':
        if keys[pygame.K_t] and self.attack_available and not self.attack_cooldown and not self.attack2_cooldown:
            self.attack_cooldown = True
            if other.inrange: # If the other character is within range
                other.take_damage(50)
                other.inrange = False
                other.is_knocked_back = True
            self.skill_counter += 1
            self.ulti_counter += 1
            self.skill_counter = min(3, self.skill_counter)
            self.ulti_counter = min(8, self.ulti_counter)
            self.attack_available = False
            self.attack_cooldown = True
            self.is_attacking = True
            character1_attack1_sound.play()
        if self.attack_available:
            if self.frame_index > 2:
                self.is_attacking = False
        if not keys[pygame.K_t]:
            self.attack_cooldown = False

    if keys[pygame.K_g] and self.attack_available and not self.attack2_cooldown:
        if self.skill_counter == 3:
            self.attacking_range = True
            self.skill_counter = 0
            self.xmovement = False
            self.ymovement = False
            character1_attack2_sound.play()

    if self.attacking_range == True:
        self.range = 'range'
        self.is_attacking2 = True
        self.attack2_cooldown = True
        if self.frame_index > 6:
            if other.inrange:
                other.take_damage(70)
                other.inrange = False
                multiplier = 5

```

Image 13: character.py: Character score and animation

```

def score(self, other): # Sets the winning condition
    if self.alive == False and not self.score_added: # Adds the score to the other character if not alive
        other.point += 1
        self.score_added = True
    if (self.point - other.point == 2 and self.point > other.point) or (self.point >= 3 and other.point != self.point): # sets who wins the game, either 2 point difference or 3+
        self.win = True
    # Sets the action for the animation
    def set_action(self, action):
        if self.current_action != action:
            self.frame_index = 0 # Reset to the first frame of the new action
            self.current_action = action
    # Update the animation of the character
    def update_animation(self, character):
        # Speed of the frames logic for each action
        if self.current_action == 'idle' or 'walk' or 'dead' or 'hurt':
            self.animation_speed = 0.01
        if self.current_action == 'run':
            self.animation_speed = 0.2
        if self.current_action == 'jump':
            if self.frame_index < 4:
                self.animation_speed = 0.7
            else:
                self.animation_speed = 0.05
        if character1.current_action == 'attack1_side' or 'attack1_up':
            self.animation_speed = 0.6
        if character1.current_action == 'attack2_side' or 'attack2_up':
            if character1.frame_index < 7:
                self.animation_speed = 0.6
            elif character1.frame_index >= 7:
                self.animation_speed = 0.1
        if character1.current_action == 'ult_side':
            if self.frame_index < 6:
                self.animation_speed = 0.6
            elif self.frame_index > 6:
                self.animation_speed = 0.02
        if character1.current_action == 'ult_up':
            if self.frame_index < 7:
                self.animation_speed = 0.6
            elif self.frame_index >= 7:
                self.animation_speed = 0.05
        if character2.current_action == 'attack1_side' or 'attack1_up':

```

Image 14: character.py: Character draw

```

def score(self, other): # Sets the winning condition
    if self.alive == False and not self.score_added: # Adds the score to the other character if not alive
        other.point += 1
        self.score_added = True
    if (self.point - other.point == 2 and self.point > other.point) or (self.point >= 3 and other.point != self.point): # sets who wins the game, either 2 point difference or 3+ points
        self.win = True
# Sets the action for the animation
def set_action(self, action):
    if self.current_action != action:
        self.frame_index = 0 # Reset to the first frame of the new action
    self.current_action = action
# Update the animation of the character
def update_animation(self, character):
    # Speed of the frames logic for each action
    if self.current_action == 'idle' or 'walk' or 'dead' or 'hurt':
        self.animation_speed = 0.01
    if self.current_action == 'run':
        self.animation_speed = 0.2
    if self.current_action == 'jump':
        if self.frame_index < 4:
            self.animation_speed = 0.7
        else:
            self.animation_speed = 0.05
    if character1.current_action == 'attack1_side' or 'attack1_up':
        self.animation_speed = 0.6
    if character1.current_action == 'attack2_side' or 'attack2_up':
        if character1.frame_index < 7:
            self.animation_speed = 0.6
        elif character1.frame_index >= 7:
            self.animation_speed = 0.1
    if character1.current_action == 'ult_side':
        if self.frame_index < 6:
            self.animation_speed = 0.6
        elif self.frame_index >= 6:
            self.animation_speed = 0.02
    if character1.current_action == 'ult_up':
        if self.frame_index < 7:
            self.animation_speed = 0.6
        elif self.frame_index >= 7:
            self.animation_speed = 0.05
    if character2.current_action == 'attack1_side' or 'attack1_up':

```

Image 15: character.py: Character reset and update

```

def score(self, other): # Sets the winning condition
    if self.alive == False and not self.score_added: # Adds the score to the other character if not alive
        other.point += 1
        self.score_added = True
    if (self.point - other.point == 2 and self.point > other.point) or (self.point >= 3 and other.point != self.point): # sets who wins the game, either 2 point difference or 3+ points
        self.win = True
# Sets the action for the animation
def set_action(self, action):
    if self.current_action != action:
        self.frame_index = 0 # Reset to the first frame of the new action
    self.current_action = action
# Update the animation of the character
def update_animation(self, character):
    # Speed of the frames logic for each action
    if self.current_action == 'idle' or 'walk' or 'dead' or 'hurt':
        self.animation_speed = 0.01
    if self.current_action == 'run':
        self.animation_speed = 0.2
    if self.current_action == 'jump':
        if self.frame_index < 4:
            self.animation_speed = 0.7
        else:
            self.animation_speed = 0.05
    if character1.current_action == 'attack1_side' or 'attack1_up':
        self.animation_speed = 0.6
    if character1.current_action == 'attack2_side' or 'attack2_up':
        if character1.frame_index < 7:
            self.animation_speed = 0.6
        elif character1.frame_index >= 7:
            self.animation_speed = 0.1
    if character1.current_action == 'ult_side':
        if self.frame_index < 6:
            self.animation_speed = 0.6
        elif self.frame_index >= 6:
            self.animation_speed = 0.02
    if character1.current_action == 'ult_up':
        if self.frame_index < 7:
            self.animation_speed = 0.6
        elif self.frame_index >= 7:
            self.animation_speed = 0.05
    if character2.current_action == 'attack1_side' or 'attack1_up':

```

Image 16: character.py: Character 1 and 2

```

# Initialize character 1
character1 = Character('character1', 1.2 * WIDTH // 3, HEIGHT // 2, 'left', animations={
    'walk': character1_walk, # List of frames for walking
    'idle': character1_idle, # List of frames for idle
    'jump': character1_jump, # List of frames for jumping
    'dead': character1_dead, # List of frames for dead
    'attack1_side': character1_attack1_side, # List of frames for attack1
    'attack1_up': character1_attack1_up,

    'attack2_side': character1_attack2_side, # List of frames for attack2
    'attack2_up': character1_attack2_up,

    'ult_side': character1_ult_side, # List of frames for ult
    'ult_up': character1_ult_up,

    'hurt': character1_hurt, # List of frames for hurt
    'run': character1_run # List of frames for run

})

# Initialize character 2
character2 = Character('character2', 1.65 * WIDTH // 3, HEIGHT // 2, 'right', animations={
    'walk': character2_walk, # List of framse for wallking
    'idle': character2_idle, # List of frames for idle
    'jump': character2_jump, # List of frames for jumping
    'dead': character2_dead, # List of frames for dead
    'attack1_side': character2_attack1_side, # List of frames for attack1
    'attack1_up': character2_attack1_up,

    'attack2_side': character2_attack2_side, # List of frames for attack2
    'attack2_up': character2_attack2_up,

    'ult_side': character2_ult_side, # List of frames for ult
    'ult_up': character2_ult_up,

    'hurt': character2_hurt, # List of frames for hurt
    'run': character2_run # List of frames for run

})

```

Image 17: hud.py: Drawing the HUD display for the characters

```

  main.py character.py hud.py x menu.py background.py
  hud.py > draw_hud
5 def draw_hud(offset_x, offset_y):
34     score_font = pygame.font.Font(None, 48)
35     hud_font = pygame.font.Font(None, 36)
36
37     # Define colors for HUD
38     health_bar_color = (255, 0, 0)
39     skill_bar_color = (0, 255, 0)
40     background_bar_color = (50, 50, 50)
41     # Bar size
42     health_bar_width = 300
43     bar_height = 40
44     max_skill_arc = 360 # Max degrees for the skill
45
46     # Character 1's attack2 circle properties
47     skill_circle_radius_1 = 30 # Radius for Character 1's attack2 counter
48     skill_circle_thickness_1 = 10 # Thickness for Character 1's attack2 circle
49     skill_circle_color_1 = (211, 211, 211)
50
51
52     ulti_circle_radius_1 = 50 # Radius for Character 1's skill counter
53     ulti_circle_thickness_1 = 20 # Thickness for Character 1's skill circle
54
55     # Character 2's attack2 skill circle properties
56     skill_circle_radius_2 = 30 # Radius for Character 2's skill counter
57     skill_circle_thickness_2 = 10 # Thickness for Character 2's skill circle
58     skill_circle_color_2 = (211, 211, 211)
59
60
61     ulti_circle_radius_2 = 50 # Radius for Character 1's skill counter
62     ulti_circle_thickness_2 = 20 # Thickness for Character 1's skill circle
63
64     # Draw Character 1's HUD
65     # Score for character 1
66
67     screen.blit(score_font.render(f"Score: {character1.point}", True, (0, 0, 255)), (WIDTH // 3 - 100, 120))
68

```

Image 18: menu.py: Showing the winning menu with play again and exit button

```

  main.py character.py hud.py x menu.py background.py
  menu.py > show_menu
4 # Function to blur the background
5 def blur_surface(surface, amount):
6     blur = pygame.transform.smoothscale(surface, (surface.get_width() // amount, surface.get_height() // amount))
7     return pygame.transform.smoothscale(blur, surface.get_size())
8
9 # Menu function to the winner and points
10 def show_menu(screen, background_image, player1_points, player2_points, winner, WIDTH, HEIGHT):
11     blurred_background = blur_surface([background_image, 10])
12     winner_font = pygame.font.Font(None, 130)
13     points_font = pygame.font.Font(None, 120)
14
15     # Determine winner text color
16     if winner == 1:
17         winner_text = winner_font.render("Player 1 Wins!", True, (255, 0, 0)) # Red
18     else:
19         winner_text = winner_font.render("Player 2 Wins!", True, (0, 0, 255)) # Blue
20
21     # Points text
22     player1_points_text = points_font.render(f"{player1_points}", True, (255, 0, 0)) # Red
23     player2_points_text = points_font.render(f"{player2_points}", True, (0, 0, 255)) # Blue
24
25     # Button size
26     button_width, button_height = 300, 80
27     play_again_rect = pygame.Rect((WIDTH // 2 - button_width // 2, HEIGHT // 2), (button_width, button_height))
28     exit_rect = pygame.Rect((WIDTH // 2 - button_width // 2, HEIGHT // 2 + 120), (button_width, button_height))
29
30     while True:
31         screen.blit(blurred_background, (0, 0)) # Draw the blurred background
32
33         # Draw the winning texts
34         screen.blit(player1_points_text, (WIDTH // 2 - winner_text.get_width() - player1_points_text.get_width() + 50, HEIGHT // 2 - 200))
35         screen.blit(winner_text, (WIDTH // 2 - winner_text.get_width() // 2, HEIGHT // 2 - 200)) # winner text
36         screen.blit(player2_points_text, (WIDTH // 2 + winner_text.get_width() - 50, HEIGHT // 2 - 200)) # Player 2 score
37
38         # Draw buttons
39         pygame.draw.rect(screen, (0, 128, 0), play_again_rect) # Green play again button
40         pygame.draw.rect(screen, (128, 0, 0), exit_rect) # Red exit button

```

Image 19: collision.py: Defining all the collisions in the map

```
main.py character.py hud.py menu.py collision.py X
collision.py > ...
1 import pygame
2
3 WIDTH, HEIGHT = 1920, 1080
4
5 # Collision class to produce the collisions for every rect and return the value
6 class Collision:
7     def __init__(self, name): # Sets the name of each rect initially
8         self.name = name
9
10    rect(self, x, y, w, h): # Creates the rect based on the values given and returning the value
11        rect = pygame.Rect(x, y, w, h)
12        return rect
13
14    # All the rects with the names and values so that it can be used to check for collision
15    ground = Collision('ground').rect((WIDTH // 3), (1.14 * HEIGHT // 2), (WIDTH // 3), 1)
16    bottom = Collision('bottom').rect((WIDTH // 3), (1.14 * HEIGHT // 2) + 1, (WIDTH // 3), (HEIGHT // 2))
17    platform1 = Collision('platform1').rect(460, 470, 275, 1)
18    platform2 = Collision('platform2').rect(715, 370, 485, 1)
19    platform3 = Collision('platform3').rect(1190, 470, 275, 1)
20    border_bottom = Collision('border_bottom').rect(-1000, HEIGHT + 1000, WIDTH + 2000, 1000)
21    border_top = Collision('border_top').rect(-1000, HEIGHT - 4000, WIDTH + 2000, 1000)
22    border_left = Collision('border_left').rect(-3000, HEIGHT - 1000, 2000, HEIGHT + 2000)
23    border_right = Collision('border_right').rect(WIDTH + 1000, -1000, 2000, HEIGHT + 2000)
24
25
26
```

Image 20: background.py: Setting the background and offsets

```
main.py character.py hud.py menu.py background.py X
background.py > ...
1 import pygame
2 from character import character1, character2 # Import the characters
3
4 WIDTH, HEIGHT = 1920, 1080
5 screen = pygame.display.set_mode((WIDTH, HEIGHT))
6 zoom_factor = 1.2
7
8 def zoom(image, zoom_factor): # Zooms the background image
9     zoomed_image = pygame.transform.scale(image, (int(WIDTH * zoom_factor), int(HEIGHT * zoom_factor)))
10    return zoomed_image
11
12 def camera(): # Sets the offset for the camera
13     init_mid_x = 1.425 * WIDTH // 3 # The initial average or midpoint of the two characters
14     init_mid_y = 0.99 * HEIGHT // 2
15
16     mid_x = (character1.xpos + character2.xpos) / 2 # Calculates the change of the average of the two characters in real time
17     mid_y = (character1.ypos + character2.ypos) / 2
18
19     offset_x = mid_x - init_mid_x # Calculates the coordinates of the midpoint where the camera should be
20     offset_y = mid_y - init_mid_y
21     offset_x = max(-200, min(200, offset_x)) # Sets the max and the min of the camera so that it doesn't exceed the background image
22     offset_y = max(-200, min(200, offset_y))
23
24     return offset_x, offset_y # Return the offsets
25
26 def center_background(zoomed_image): # Center the background initially
27     bg_width, bg_height = zoomed_image.get_size()
28     x_offset = (WIDTH - bg_width) // 2
29     y_offset = (HEIGHT - bg_height) // 2
30
31     # Return the live offset values along with the offsets
32     return x_offset, y_offset
33
```

Final Result:

