BOĞAZİÇİ UNIVERSITY

Faculty Of Industrial Engineering

IE201: Object Oriented

Programming

Instructor: Zeki Caner Taşkın

2023 - Fall Term

27.12.2023

Project Part 3 - Final

Selahattin Eyüp Gülçimen

Melisa Akansel

Duhan Yurttaş

Enes Pekalkan

Table Of Contents:

1. Game Introduction

Harry Potter: Hogwarts Under Attack

Overview:

Embark on an epic journey in this Harry Potter-themed 2D platform game. Your mission is clear: eliminate dementors, confront Voldemort, safeguard Hogwarts, and emerge victorious!

Platforms & Navigation:

The game features three vertical platforms.

Navigate Harry Potter across these platforms, switching between them as needed.

Combat Mechanics:

Equip Harry with his wand to cast spells (activated by pressing key 'X').

Each dementor requires one hit of magic to defeat.

Beware! Dementors will randomly spawn across the platforms, challenging your combat skills.

Health & Damage:

Harry starts with a health bar.

Sustaining damage from dementor attacks reduces the health bar by 25 points per hit.

If Harry's health bar depletes to zero, the game concludes with Hogwarts falling under threat.

Potions & Power-ups:

Discover random potions scattered throughout the platforms.

Collecting a potion replenishes Harry's health bar by 50 points, enhancing his survival chances.

Final Confrontation:

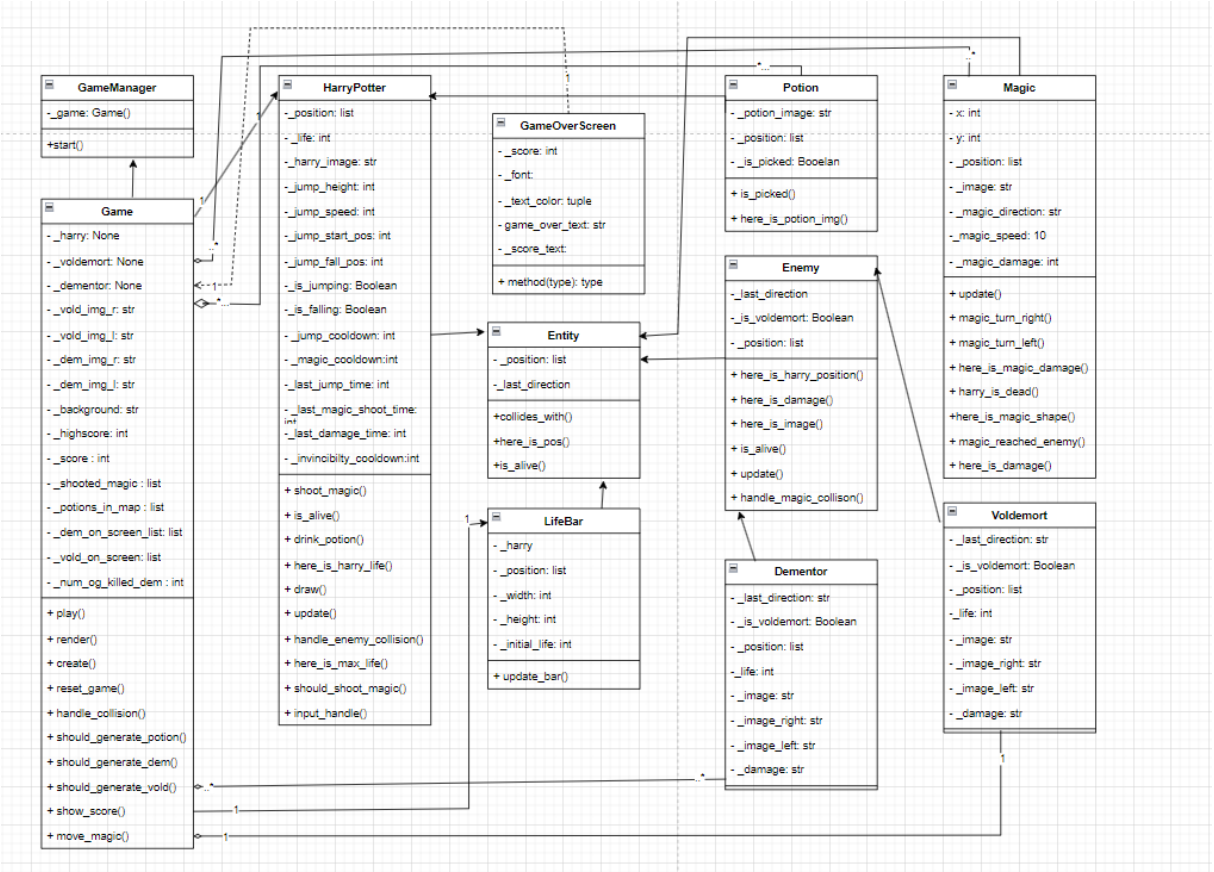
After defeating 10 dementors (a count displayed on-screen), prepare for the ultimate challenge.

The formidable Lord Voldemort emerges as the final boss many times after the 10 striked dementors, requiring strategic gameplay and skillful dodging to defeat.

Objective:

Navigate the challenges, gather potions, eliminate dementors, and face Voldemorts head-on. Your ultimate goal: save Hogwarts and ensure its safety from the dark forces once and for all!

2. Class Diagram



3. Brief Explanation Of Class Diagram

Our Class Diagram has a total of 11 classes: Game Magic, LifeBar, Entity, Enemy, Potion, Dementor, Voldemort, Harry Potter, GameOverScreen and GameManager.

Game Class:

Game class is the most important class of our diagram. This is where the game will be runned with play() function. Game has “has a” relationship with Potion, Enemy and Harry Potter classes. These are components of the game, our game involves these elements. However, Harry Potter, Enemy and Potion can exist without the game’s play itself. They are seperate entities. Therefore they are aggregated to our game class. There will be one Harry Potter in

the game and 0 potions initially. There will be 0 enemies eventually and the numbers on these aggregation lines indicate that. Game is associated with LifeBar class because LifeBar is updated with the changes that is happening in the Game class. Needs information from there. And wand is also associated with the game class since the damage will be controlled there. Game has voldemort, harry potter and dementor attributes as none. These entities are none type class attributes of game. Their images and relative positions according to right and left directions are defined as string and potions, voldemort, dementor and the moving magic on the map is kept track of in the map by their respective lists in the game attributes. Game has render() function to reuse some map elements when the game is started over. With create() function, living components of our game harry and enemies are created. Game is also responsible for checking the generation of potion, dementor and voldemort should_generate() as well as keeping and showing the score of the player by show_score().

LifeBar Class:

For Harry's health we are defining a Lifebar class. Harry Potter has a lifebar therefore it is aggregated to Harry Potter class and associated with game as we discussed above. It'll be shown on screen so it has width and height attributes. It has only one function update_bar() which acts according to enemy collisions or potion obtaining.

Entity Class:

To make coding easier for our dynamic entities, we are defining an entity class which will check their positions collision and aliveness for our moving elements: Harry Potter, Enemies and Magic. position attribute is useful for keeping the coordinates of our moving entities on the gamescreen. Last direction attribute is useful for updating the images of entities accordingly. Collides_with() function checks whether any collision occurred among any of these 2 entities and here_is_pos() gives current position of one entitiy to another to continiously check for collisions. is_alive() function checks the living status of entities to whether remove magic and dementors from the screen or to end the game if harry is dead.

Magic Class:

For the magic Harry Potter throws to deal magic we are defining a wand class. It is associated with game on both sides since its position on the map will be updated and the damage will be dealt accordingly they both need information from each other. Magic has attributes like speed and damage as integers and a list that keeps where the magic is currently on the map to

control its movement. Functions like magic_reached_enemy and here_is_damage() deals with the collision effect of the magic. Magic's direction is determined by magic_turn_left() magic_turn_right() functions and so on and so forth.

Enemy Class:

For the enemies Harry is going to face we are creating an enemy class, with two subclasses that inherits from enemy class, which are different enemy types: **Dementor Class** and **Voldemort Class.** Enemy is associated with wand since it's death is dependent on the wand and the magic it throws on the map. Harry Potter is also associated with enemy class because he might get damage from the collision. Dementor is the common enemy while Voldemort is the final boss.

Enemy has a boolean attribute is_Voldemort if this value returns true that means the enemy is voldemort and voldemort is spawned its condition is killed_dem_count>=10 Enemy class also has an attribute as list which keeps the positions of existing enemies on the map. Enemy has functions that gives its damage, image (here_is_damage(), here_is_image()) and handle_magic_collision() which checks the whether the magic hit the enemy or not.

Potion Class:

For obtainable health potions we are defining a potion class. It is associated with Harry Potter since the picked potions increase harry's lifebar, needs information from Harry to update the lifebar. And aggregated with game class as discussed above. Potion has a list attribute that keeps potions positions on the map and the image attribute as a string. if is_picked = Boolean attribute returns true that means the potion is obtained and harry's lifebar is increased by +50 accordingly. here_is_potion_image() function gives its image which was defined as its attribute earlier.

GameOverScreen Class:

This class serves a crucial role in managing the display of the game over screen and facilitating the option to restart the game. It is directly associated with the main game class. It has attributes such as score, font, text_color, score_text. And 2 functions render() which generates the map again with food and enemies and

GameManager Class:

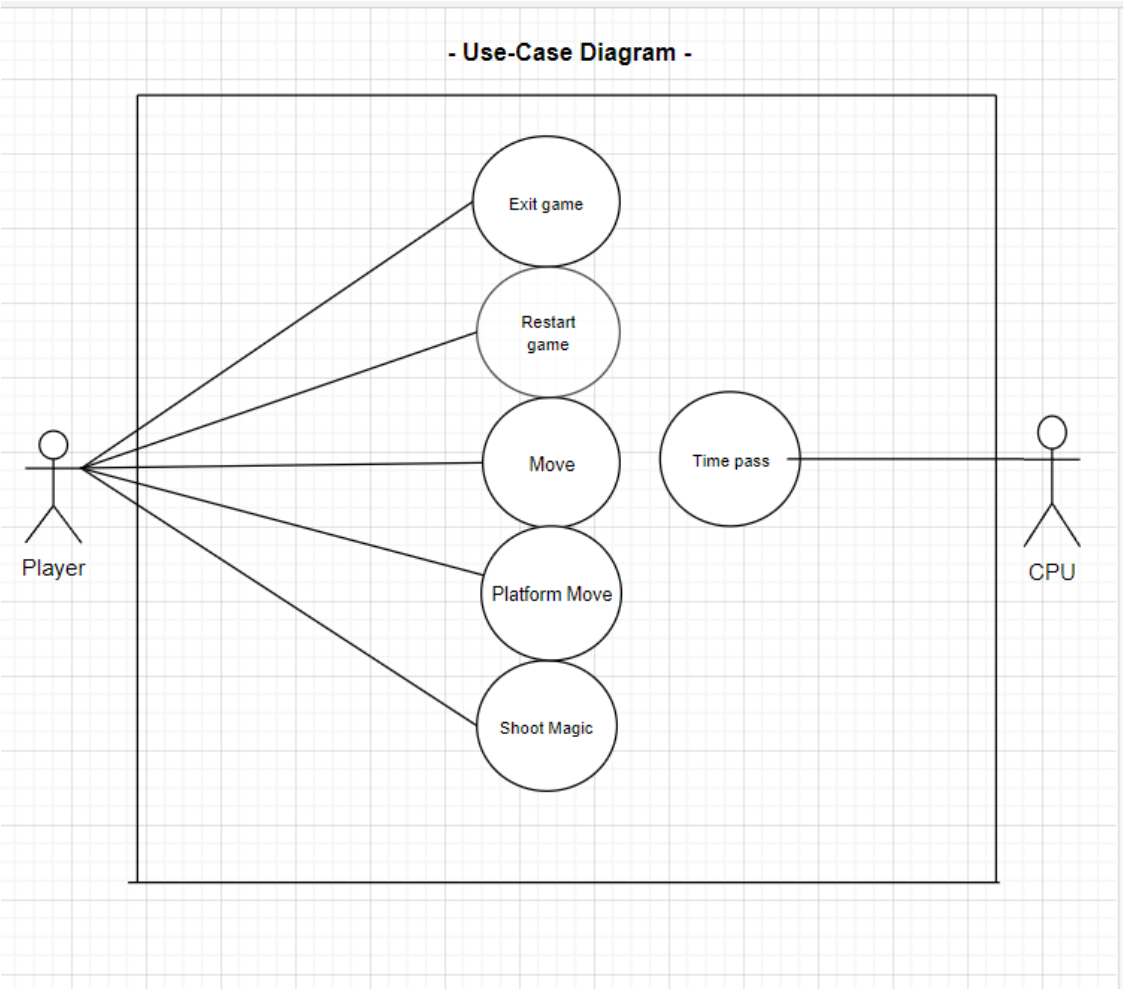
This class is only responsible of initiating and starting the game. It's associated with game

class. It only has 1 function which is starts the game start().

HarryPotter Class:

HarryPotter is the playable character in the game. It has many attributes, such as magic and jump cool down to prevent spamming a button too much. LifeBar value as integer and a list that keeps his position. To calculate the cooldown time we also defined the last magic shoot times and last jumping time to know the interval. Harry has shoot_magic() function which makes him shoot magic with spaceBar. drink_potion() function that helps him increase his lifebar. Input_handle() function that controls his movement around the map and many more functions.

4. Use Case Diagram



5. Brief Explanation Of Use Case Diagram

Our use cases are, events that are directly triggered by the users engagement with keys. We have 5 use cases in our diagram.

Restart Game Use Case:

To restart the game users can engage with the keySPACE.

Exit Game Use Case:

To exit the game users can engage with the mouse button down and pygame.Quit is realized.

Shoot Magic Use Case:

To shoot magic users can engage with the key.SPACE

Move Use Case:

To move left users can engage with the key.LEFT and to move right users can engage with the key.RIGHT

Platform Move Use Case:

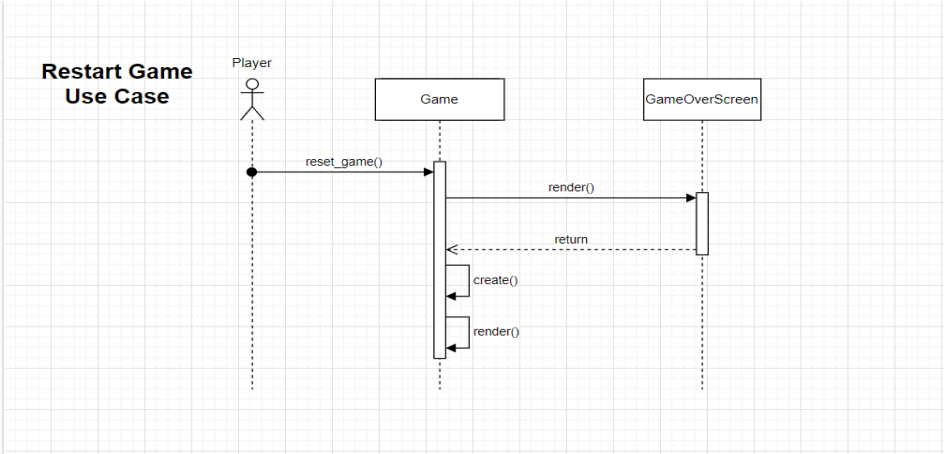
To move to the upper platform users can engage with the key.UP. To move to the lower platform users can engage with the key.DOWN

6. Collabration And Sequence Diagrams

Collaboration diagrams offer a clear picture of how the code functions when a user interacts with defined use-cases. Essentially, they illustrate the sequence of actions that unfold in the code once a command is initiated. In these diagrams, classes interact by exchanging parameters through their functions. These parameters essentially translate into messages between different class instances. By observing a collaboration diagram, we can trace the flow of operations within the code. On the contrary, a sequence diagram provides insights into the dynamic behavior of the system by showcasing the chronological order of interactions among different components or objects in a system. While collaboration diagrams emphasize the interactions between classes and their methods, sequence diagrams focus more on the timing and order of these interactions

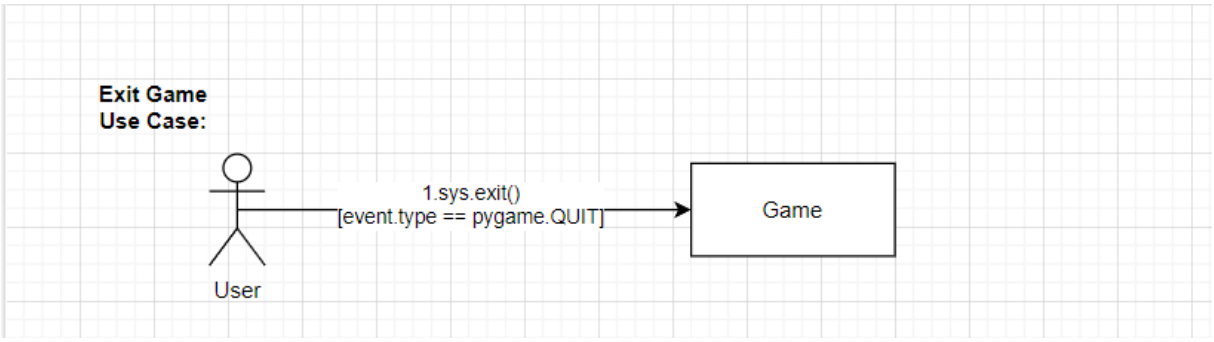
a) Restart Game:

When the game is over user can press the key.SPACE, then restart_game() function would give the restart message to game. render() function sends the message to GameOverScreen (1.1.render()). And creates the map again (1.3 create()). Then render() function tells the Game to render the map elements like enemies, player healthbar etc again (1.4. render()).



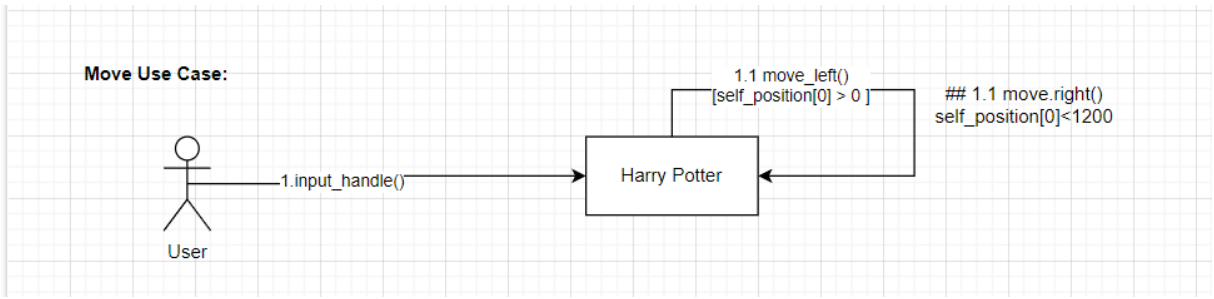
b) Exit Game:

When the user clicks on quit button of the window at the upper right corner, sys_exit() function sends the quitting message to the game and game closes itself.



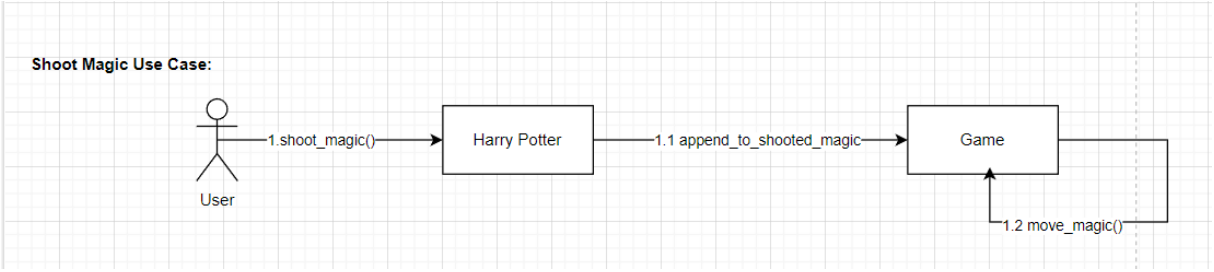
c) Move:

User engages with the key and input_handle() function sends the message to player. move() functions sends the message again to the player instance which is our character in the game and player moves accordingly. For instance, if the key.RIGHT input is taken, handle_input() function takes the user input decides the action and sends the message to the player to take action. move_right() function sends this message back to player and moves the player to the right



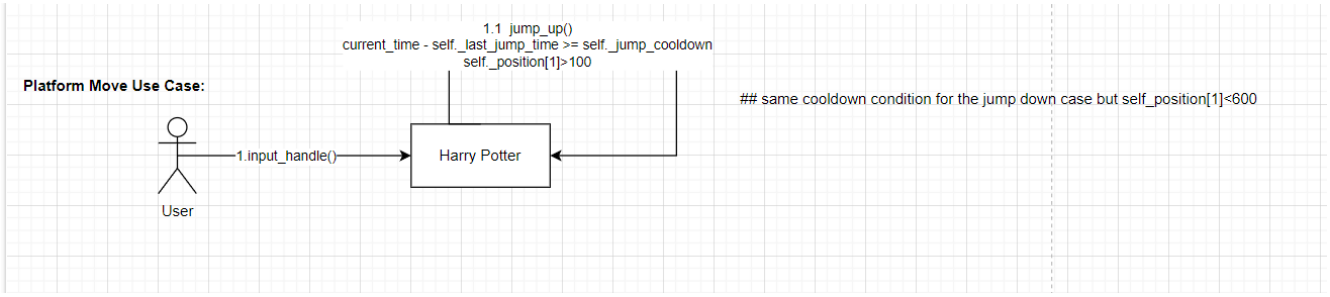
d) Shoot Magic:

User engages with the key and shoot_magic() function takes the input message and sends it to the Harry Potter. Harry Potter then sends the message that a magic is shot to the game class and game appends this magic to the list of shooted_magic. Then game class sends another message to itself to update the appended magic's position and movement on the map with move_magic().



e) Platform Move:

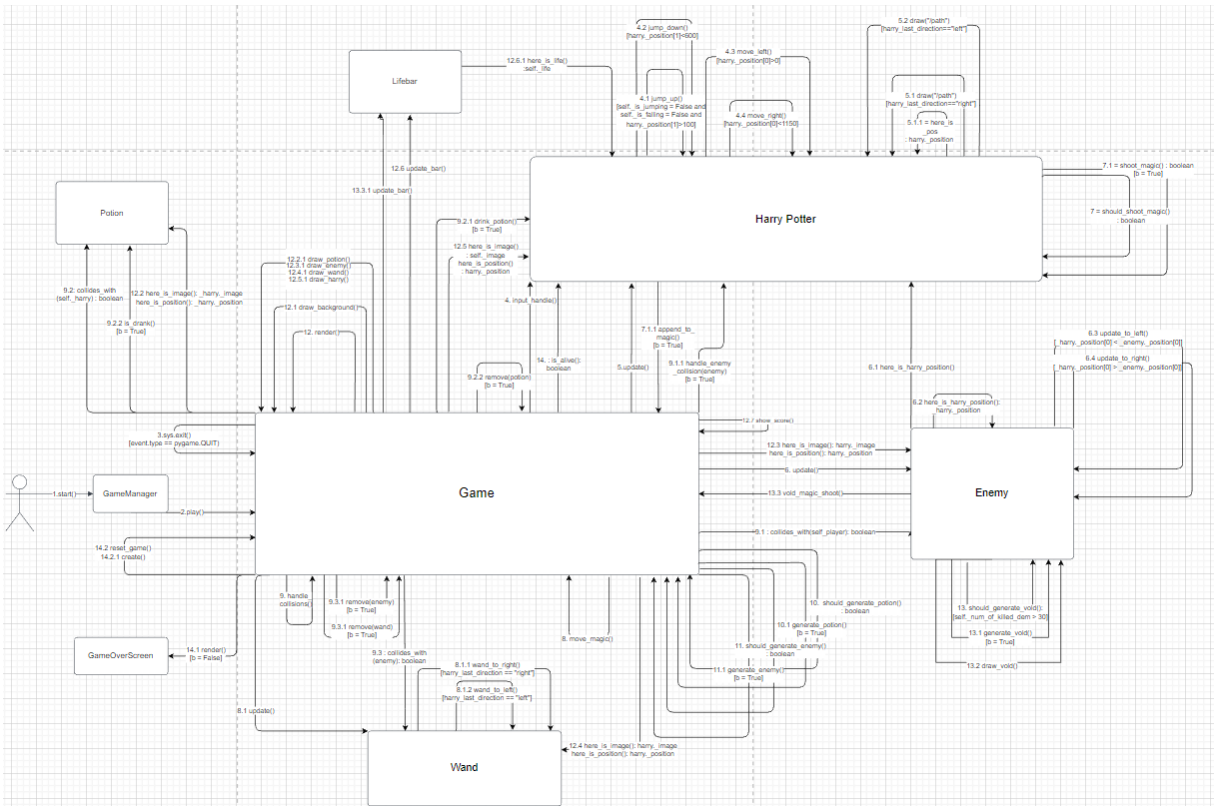
For vertical movements we also added cooldown time. Hence jumping inputs also check the cooldown time apart from the initial position of character.



f) Tick:

First of all, 1.start() function starts running as soon as the game is opened this function sends a message to GameManager to start the game. This message triggers 2.play() function in the game. Which is where the while loop of the whole game is constructed, and runned until the death of harry. 3.sys.exit() is a built-in method of pygame, used for terminating the game screen. 4.input_handle() is where the user commands for our playable character harry potter is realized. 4.1 jump_up(), 4.2 jump_down() functions control the platform moves we defined earlier with certain conditions. These conditions are given to prevent from going lower than the base platform and upper than the highest platform. 4.3 and 4.4 move_right and move_left() functions control the left and right movements of harry potter, also with certain conditions to prevent from going further from the borders of the screen. 5.update() updates Harry Potter's image left or right according to the input that was handled in the 4.input_handle().6.update() updates enemy entity's moves . Enemy keeps the harry._position attribute all the time to check for collisions. 6.1 sends this information to harry potter class and 6.2 keeps the information for the enemy class. 6.3 and 6.4 updates enemy direction

according to the direction harry potter is headed to. 7.should_shoot_magic() checks whether the input of shooting magic is taken or not 7.1 shoot_magic, conducts the operation if the should_shoot_magic() functions value, a boolean is true. Then according to this shoot, 7.1.1 append_to _magic keeps track of the magic on the game map by puting it on a list. Since the magic is shot and appended to the list on a game to, keep track of it. 8.move_magic() moves the position of the shooted magic with the given speed accordingly. 8.1 updates wand. 8.1.1 and 8.1.2 updates the direction of the magic headed to according to the direction of harry. 9.handle_collisions(), checks the collisions between our entities or game elements. For example 9.2 collides_with() checks whether the potion and harry's positions are in the same place. If the boolean returns true then the 9.2.1 drink_potion() function of harry potter is = True then 9.2.2 remove(potion) sends the message to itself of game and removes the such potion from the game map. 10.should_generate_potion() and 11.should_generate_enemy() checks if the game's condition to generating this elements are satisfied is yes boolean returns true and 10.1 generate_potion() and 11.1 generate_enemy() is realized. 12 render() render's all game elements again when the game is started over again. For instance 12.5 here_is_harry_image takes harry's image and 12.5.1 draw_harry() draw's the character on the game screen with the provided image attribute of Harry Potter Classs. 13.should_generate_vold() checks whether the certain killing condition of dementors is satisfied. 13.1 generate_vold() and 13.2 draw_vold sends the message over the enemy class. 14.is_alive() checks whether the harry is still alive if not, 14.1 render() shows the game over screen. 14.2.reset_game() resets the game and 14.2.1.render() creates the game elements accordingly.


Lifebar
Potion
Harry Potter
Game
Enemy
Wand
GameManager
GameOverScreen
4.2 jump_down()
[harry_position[1]<500]
4.3 move_left()
[harry_position[0]>0]
5.2 draw("path")
[harry_last_direction=="left"]
12.6.1 here_is_life()
:self._life
4.1 jump_up()
[self._is_jumping = False and
self._is_falling = False and
harry_position[1]>100]
4.4 move_right()
[harry_position[0]<1150]
5.1 draw("path")
[harry_last_direction=="right"]
5.1.1 = here_is
_pos
:harry_position
12.6 update_bar()
13.3.1 update_bar()
7.1 = shoot_magic() : boolean
[b = True]
7 = should_shoot_magic()
: boolean
9.2.1 drink_potion()
[b = True]
12.5 here_is_image()
: self._image
here_is_position()
: harry_position
12.2.1 draw_potion()
12.3.1 draw_enemy()
12.4.1 draw_wand()
12.5.1 draw_harry()
9.2: collides_with
(self._harry) : boolean
12.2 here_is_image() : harry_image
here_is_position() : harry_position
9.2.2 is_drank()
[b = True]
12.1 draw_background()
12. render()
4. input_handle()
7.1.1 append_to
_magic()
[b = True]
14. : is_alive():
boolean
9.2.2 remove(potion)
[b = True]
5.update()
9.1.1 handle_enemy
_collision(enemy)
[b = True]
6.3 update_to_left()
[harry_position[0] < : enemy_position[0]]
6.4 update_to_right()
[harry_position[0] > : enemy_position[0]]
6.1 here_is_harry_position()
6.2 here_is_harry_position():
harry_position
12.1 show_score()
12.3 here_is_image(): harry_image
here_is_position(): harry_position
5. update()
13.3 void_magic_shoot()
9.1 : collides_with(self_player): boolean
3.sys.exit()
[event.type == pygame.QUIT]
1.start()
2.play()
14.2 reset_game()
14.2.1 create()
9. handle
collisions()
9.3.1 remove(enemy)
[b = True]
9.3.1 remove(wand)
[b = True]
9.3 : collides_with
(enemy): boolean
8. move_magic()
10. should_generate_potion()
: boolean
10.1 generate_potion()
[b = True]
11. should_generate_enemy()
: boolean
11.1 generate_enemy()
[b = True]
13. should_generate_void():
[self._num_of_killed_dem > 30]
13.1 generate_void()
[b = True]
13.2 draw_void()
14.1 render()
[b = False]
8.1.1 wand_to_right()
[harry_last_direction == "right"]
8.1.2 wand_to_left()
[harry_last_direction == "left"]
8.1 update()
12.4 here_is_image(): harry_image
here_is_position(): harry_position

7. User Manual

The arrow keys are responsible for movement:

- The left arrow key allows movement left.
- The right arrow key allows movement to the right.
- The up arrow key allows to go up to a higher platform, if available.
- The down arrow key allows to land on a lower platform, if available.

The Space key allows to throw spells.

Game can be closed by pressing the cross button on the top right.

Inıtially, game starts automatically. After the game is over, you can play it again by pressing the space bar key.