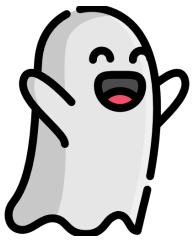


KREKHED

An Infinite Adventure

Created by Danyal Abbas

1. Overview

"Krekhed" is an infinite looping game developed by Danyal Abbas using Python and the Pygame library. Drawing inspiration from the Google Chrome dinosaur game, Krekhed presents players with a pixelated adventure set in a dynamically generated digital landscape. The game is designed as a project for "Bano Qabil" and showcases the creative and technical skills of the developer.

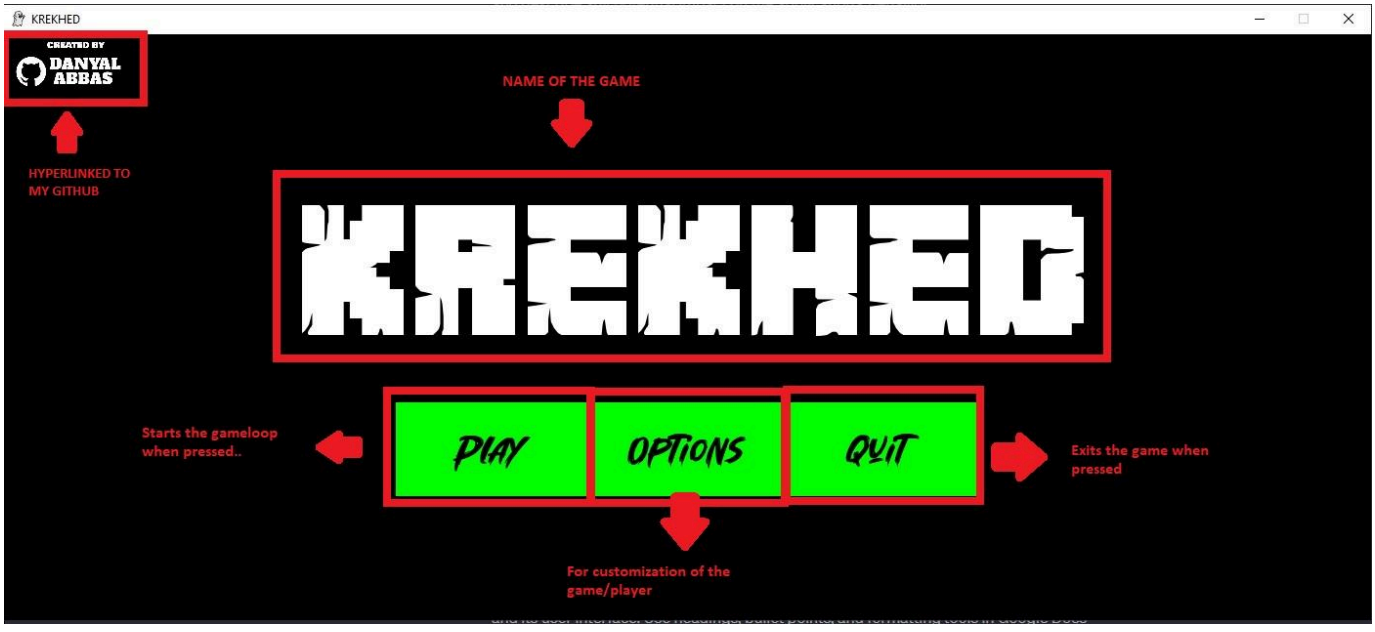
Key Features:

1. **Endless Adventure:** Krekhed offers an infinite looping gameplay experience, challenging players to navigate through dynamically generated obstacles and hurdles.
2. **Python and Pygame:** Developed with Python, a versatile programming language, and Pygame, a popular library for game development
3. **Innovation and Inspiration:** While inspired by the Google Chrome dinosaur game, Krekhed introduces innovative elements and a unique flavor, providing players with a fresh and enjoyable gaming experience.
4. **Color Customization Feature:** The options menu now includes a color customization feature, enabling players to change the appearance of Krek.
5. **Simple Controls:** The game continues to feature intuitive and easy-to-master controls, allowing players to focus on the thrill of the adventure.

2. Usage

- ## MAIN MENU SCREEN

Upon running the application, users will be greeted with a graphical user interface (GUI) Main Menu screen that consists of the name of the game (KREKHED), a “Created by Danyal” image on the top-left which will lead you to my GitHub profile upon clicking and three primary buttons: Play, Options, and Quit.



- ## GAMELOOP SCREEN

Upon clicking the Play button, users enter an infinite scrolling background game with the following features:

Player Controls:

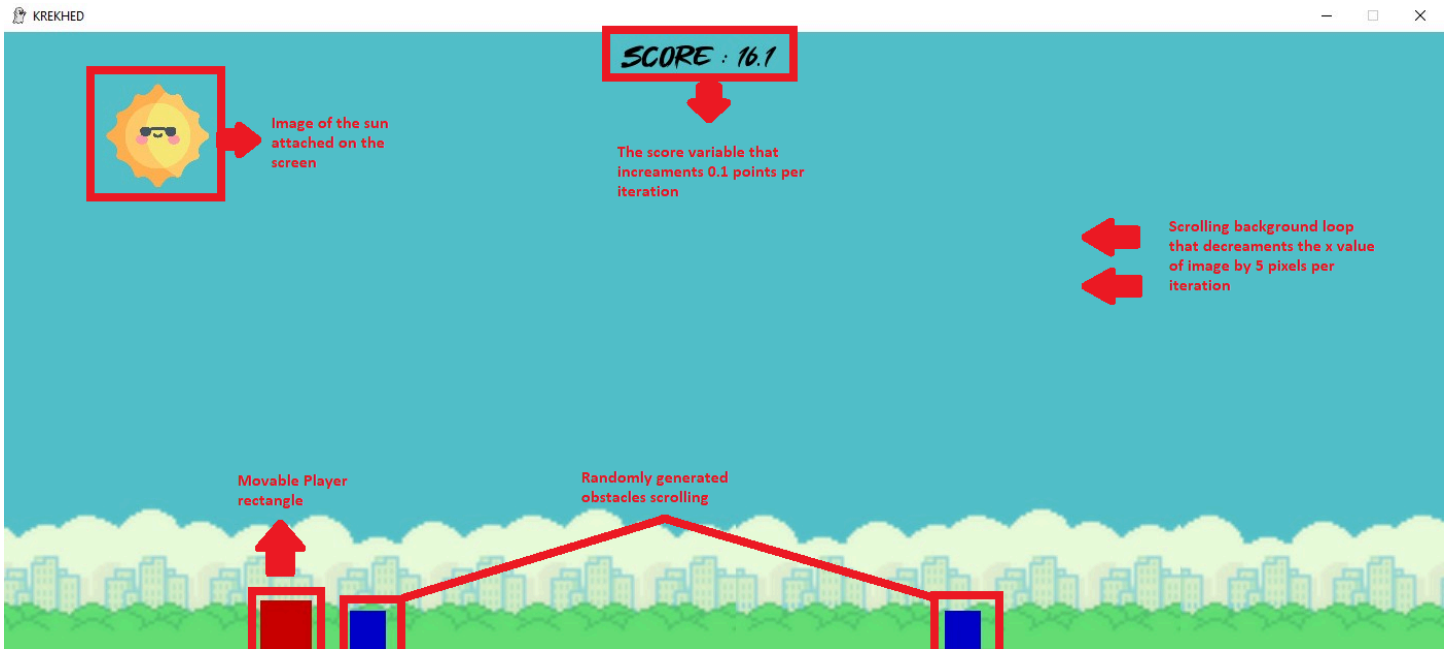
- Move Left: "A" key or left arrow key.
- Move Right: "D" key or right arrow key.
- Jump: Spacebar key.

Game Interface:

- Score Display: Top-middle of the screen.
- Player Character: Rectangle, movable left, right, and jump.
- Obstacles: Scroll continuously from right to left.

Gameplay Mechanics:

- Scoring: Increase by avoiding obstacles.
- Obstacles: Continuous scrolling; must be avoided.
- Collision: The game stops on collision; transition to the "You Died" screen.



- ## DIED SCREEN

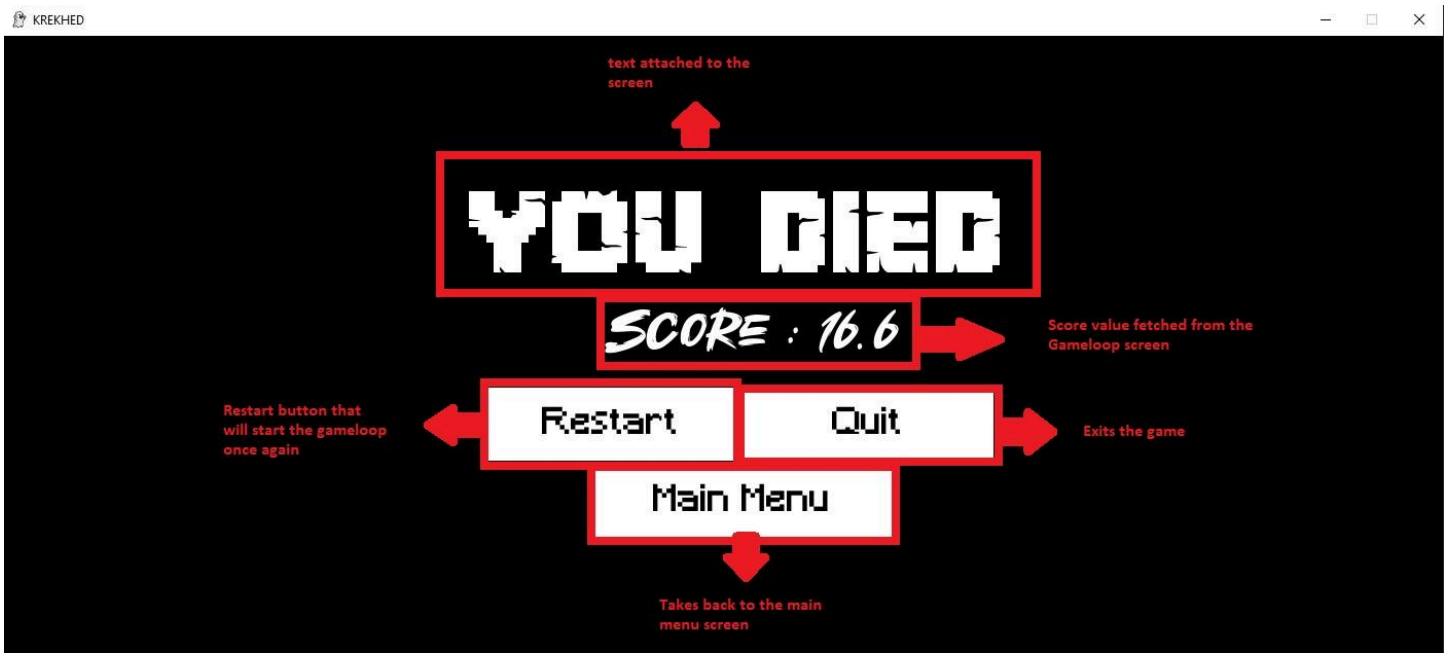
After a collision with an obstacle, the game transitions to a "You Died" screen with:

Text Display:

- "YOU DIED" at the center.
- Final Score.

Action Buttons:

- Restart: Begin a new game.
- Quit: Exit the game.
- Main Menu: Return to the main menu.



- ## OPTIONS SCREEN

Upon clicking the “Options” button, the Options Screen allows users to customize their experience with the following features:

Player Color:

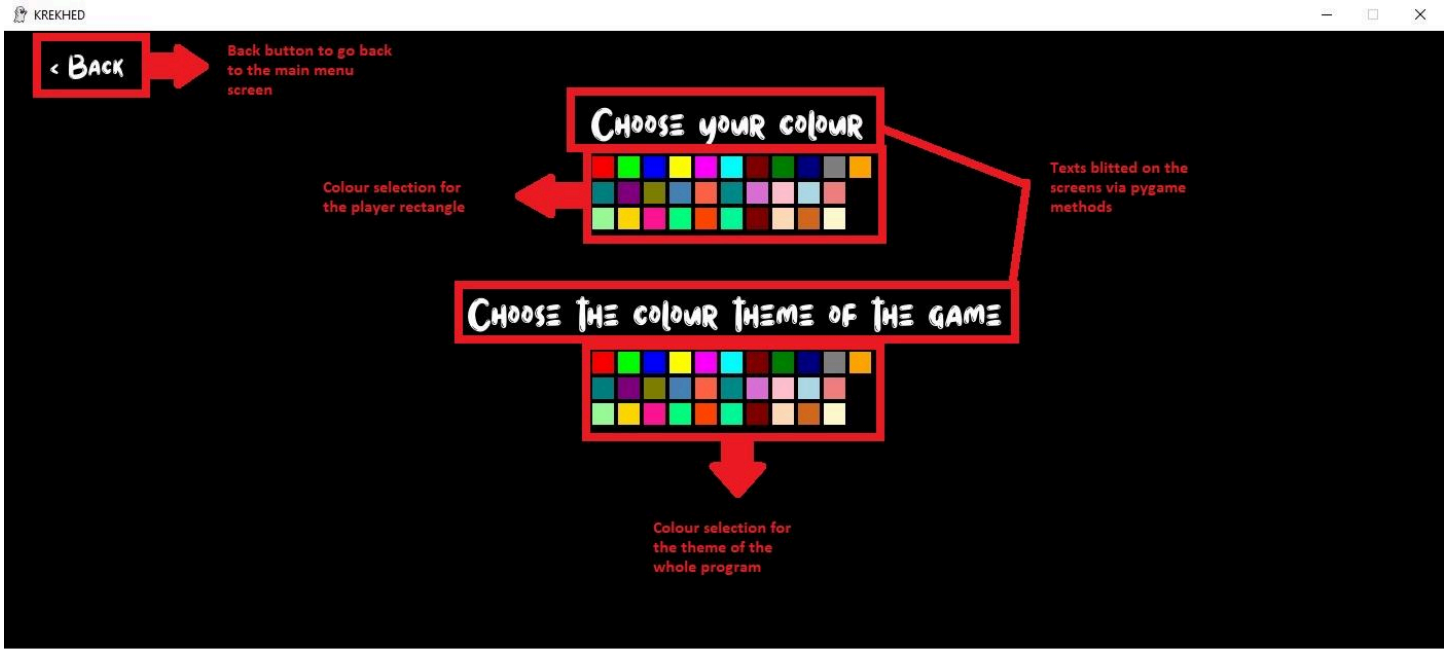
- Tiny color buttons for selecting the color of the player rectangle.

Theme Color:

- Another set of tiny color buttons for choosing the overall theme color of the program.

Back Button:

- A "Back" button to return to the main menu.



3. Code Structure

The code of the game consists of three main Classes, which are;

Krekhead(): This is the class that deals with all the normal tasks of the game i.e its methods include; Initializing pygame, setting up its resolution, scrolling background in the gameloop, creating button, and rendering text to attach to program/game

Krek(): This is the class that deals with the business related to the player and obstacles i.e, its methods include; creating and drawing the player on screen, moving the player (left/right), making the player jump, creating obstacles, moving obstacles, and the collision between the player and obstacle

Screens(): This is the class that deals with the different screens of the game and what tasks the screens will perform i.e, its methods include; DiedScreen, MainMenu, OptionsScreen, Gameloop

4. Detailed Functionality

- **PLAYER MOVEMENTS**

Player can be moved left and right by using increamenting:

First of all, to detect if certain key on keyboard are pressed or not, we use pygame methods (pygame.KEYDOWN, pygame.KEYUP) and loops to iterate through every possible event and make the values of the variables assigned to the movements True when the certain key assigned to perform a certain task is pressed

```
for event in pygame.event.get():
            if event.type == pygame.QUIT:
                sys.exit()


            if event.type == pygame.KEYDOWN:
                if event.key == pygame.K_LEFT or event.key == pygame.K_a:
                    player.move_left = True
                elif event.key == pygame.K_RIGHT or event.key == pygame.K_d:
                    player.move_right = True
                elif event.key == pygame.K_SPACE:
                    player.isJump = True
```

Left Motion:

- Press "A" key or left arrow key to move the player character left.
- The x-value of the player is decreased.
- determine if the key has been pressed by using pygame.K_a or pygame.K_left:
- we will also add borders as we do not want our player to leave the screen by using if condition

```
if player.move_left and player.x > 10:
            player.move(-6, 0)
```

Right Motion:

- Press "D" key or right arrow key to move the player character right.
- The x-value of the player is increased.
- determine if the key has been pressed by using pygame.K_a or pygame.K_left
- we will also add borders as we do not want our player to leave the screen by using if condition

```
elif player.move_right and player.x < krek.window_width - 60:
            player.move(6, 0)
```

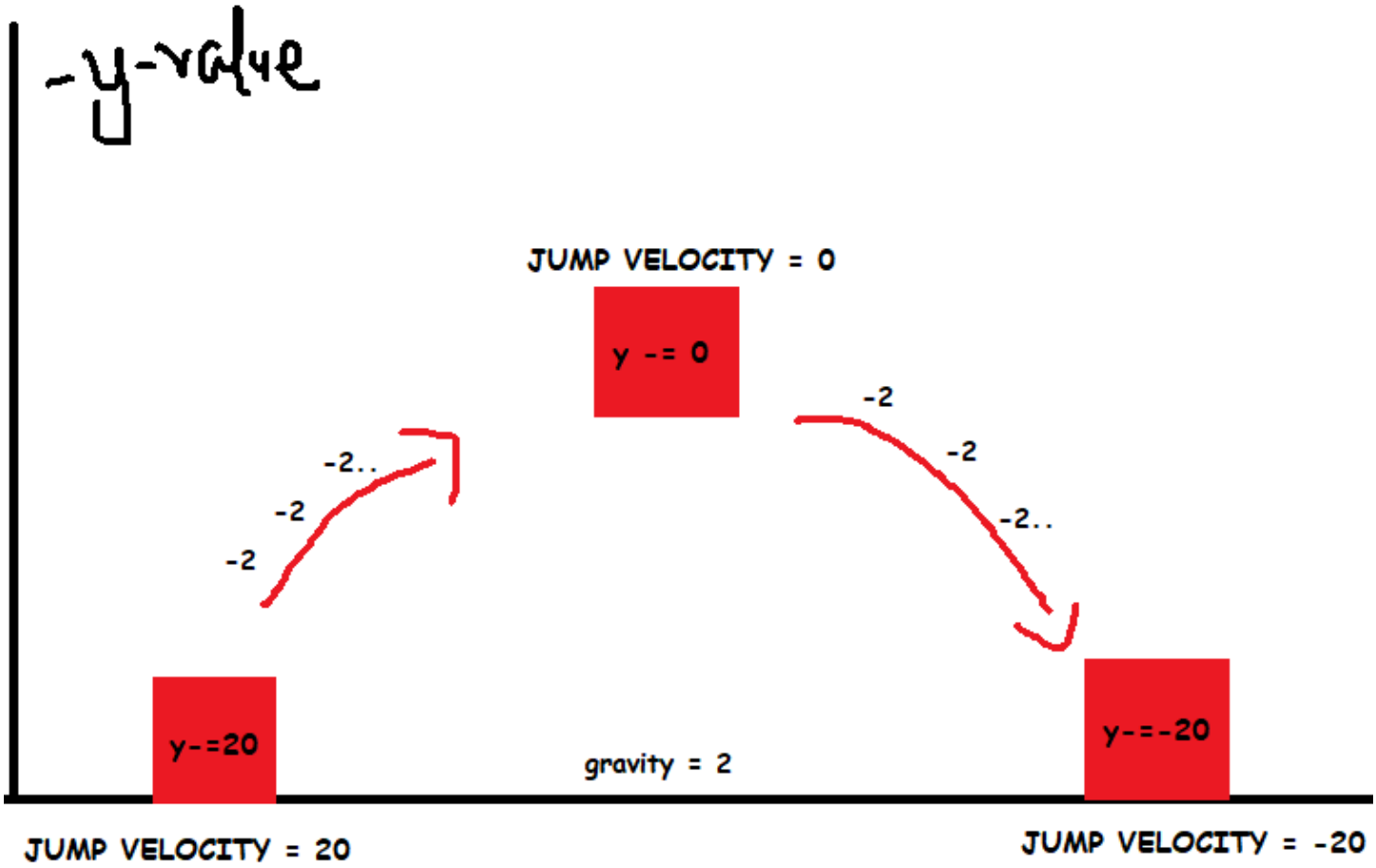
Jump:

- Press the spacebar key to trigger the jump function.

to make the player jump and make it look like the player is jumping and having certain air-time, we need to add velocity variables aswell as gravity variables.

```
        self.jump_height = 20
        self.gravity = 2
        self.jump_velocity = self.jump_height
```

we will initially assign self.jump_velocity the value of self.jump_height where jump height here is the maximum height the player can jump to, then we define a method in Krek() class named jump() that doesn't take any arguments, and when that function is called it decrease the y-value of the player by jump velocity and decrease the value of jump velocity by gravity and then we apply a condition if the absolute value of jump velocity is greater than jump height, we stop the decrement and make the jump velocity value same as jump height once again

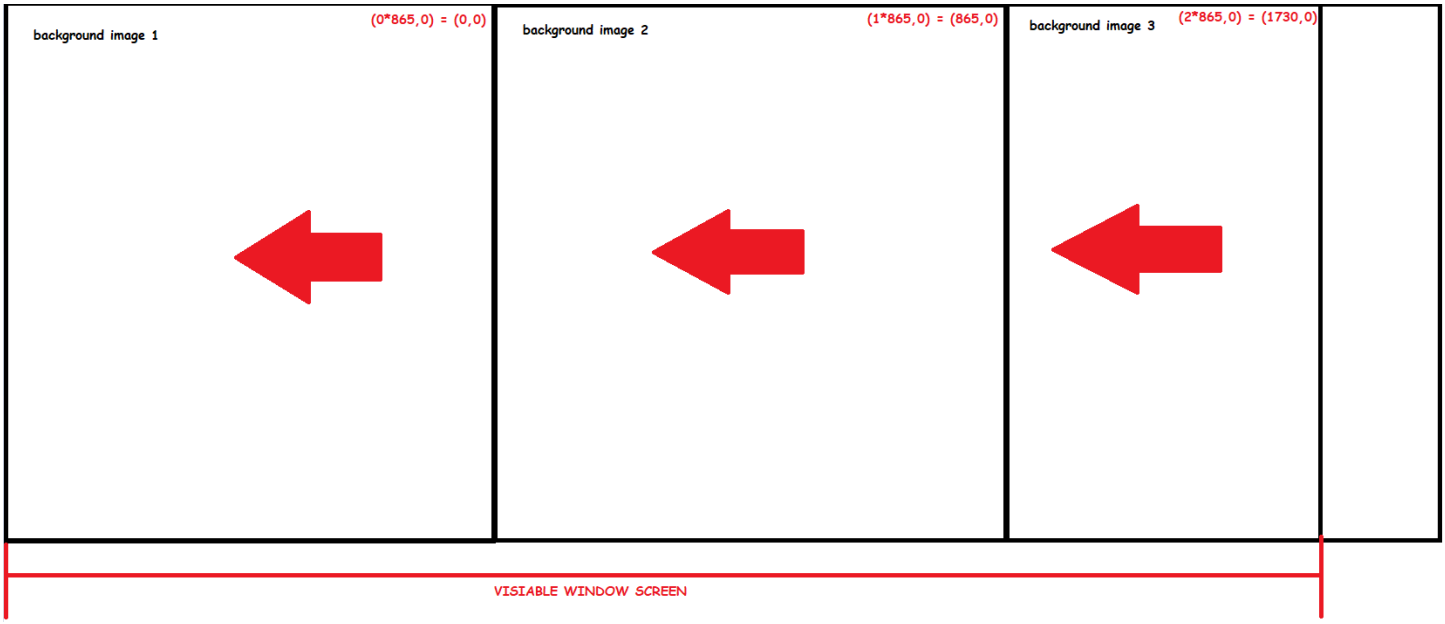



```
def jump(self):
        self.y -= self.jump_velocity
        self.jump_velocity -= self.gravity
        if abs(self.jump_velocity) > self.jump_height:
            self.isJump = False
            self.jump_velocity = self.jump_height
```

- ## SCROLLING BACKGROUND

To make infinitely scrolling background that is memory efficient aswell, we first make a method in Krekhead() Class named scroll_thingey() that we load the image we want to use for the background of our game by using pygame method pygame.image.load(***image_name.extension***) then we will get the width of the image by using another pygame function .get_width() and then we will create a variable “tiles” in which will divide the width of the window (in our case, 1400) screen with image width (in our case, 865) and as the answer can be a floating point, we use math.ceil function to round it up to nearest higher number

Then we create a for-loop that iterates from range 0 to tiles and we use blit function (pygame function to attach a image/text to the screen) and attach a background image every iteration by putting the x-value as (i * bg_width, 0), however in this case the background will not be scrolling as we are not decreasing the x value of the images, so we create a variable scroll and decrease its value by scroll -= , so now the new equation for x-value becomes (i * bg_width + scroll, 0) then we apply a condition determining if the image is fully out of the screen we assign scroll = 0 again so the scroll restarts




```
def scroll_thingey(self):
        background = pygame.image.load("Assets/bg.png").convert_alpha()
        bg_width = background.get_width()
        tiles = math.ceil((self.window_width / bg_width)) + 1

        for i in range(0, tiles):
            win.blit(background, (i * bg_width + self.scroll, 0))
        self.scroll -= 5
        if abs(self.scroll) > bg_width:
            self.scroll = 0
```

- ## OBSTACLES CREATION, DRAWING AND MOVEMENT

To create obstacles, we define a create_obstacle() function in our Krek() Class and in this function, we use randint method of the “random” library to create enemies at random positions from x-value 1400 to 2000, as we want it seem like the obstacles are coming from other side of screen. And for y-value, we assign it 560 as that is the floor for our game resolution and then create a rectangle using x.y values assigned and by using pygame function, pygame.Rect()

```
def create_obstacle(self):
        x = random.randint(1400, 2000)
        y = 560
        obstacle_rect = pygame.Rect(x, y, 35, 60)
        return obstacle_rect
```

Now for the drawing of the obstacle on the Gameloop window screen, we create another function in the Krek() Class named draw_obstacle(), which takes screen of the program as input. In this function we implement a for-loop on a list named obstacles (currently empty) use to pygame method pygame.draw.rect() to draw the obstacle rectangles on the screen window

```
def draw_obstacles(self, win):
        for i in self.obstacles:
            pygame.draw.rect(win, (0, 0, 200), i)
```

and now for the movement of the obstacles, we create another function in the Krek() Class named move_obstacles, which is assigned a default value of 10, that is the speed at which the obstacles will be scrolling through and that can be altered further through in the game. We use a for-loop on the list named obstacles and decrease the x-value of the individual obstacles by value (which, in our case is 10)

```
def move_obstacles(self, value = 10):
        for i in self.obstacles:
            i.x -= value
```

now to attach and use all these functions in the main Gameloop of the game, we go to our Gameloop() method in the Screens() Class. In the Gameloop method, we use a condition using randint function that if the randomly generated number from 0 to 320 is less than 5, than we call the create_obstacle() function, and append it into the obstacle list, and then get out of the if-condition, and use list comprehension on the obstacle list and re-append every obstacle that’s x-value is greater than -40, as the obstacles gone out of the screen would be no use to us and would take memory resulting in very slow and laggy game.

```
 if random.randint(0, 320) < 5:
     player.obstacles.append(player.create_obstacle())

 player.obstacles = [obstacle for obstacle in player.obstacles if obstacle.x > -40]
```

- ## COLLISION DETECTION

To find out if any of the obstacle rect is colliding with the player rect, we first create another method in the Krek() Class named is_collision(), in this method, we apply a for-loop in the obstacle list and we simply just use the distance formula to find the distance between the player and the (i)th obstacle.

$$d = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

here,

- x_2 is the x-value of the obstacle.
- x_1 is the x-value of the player.
- y_2 is the y-value of the obstacle.
- y_1 is the y-value of the player.

then we use an if-condition to detect, if the distance between the player and obstacle is less than a certain value (in our case, 40), and if the condition is met, we return True, else we return False.

Now to implement the following method in our Main Gameloop, we go in our Gameloop method in the Screens() Class and we use an if-condition to see if the is_collision() function is True or not, if the condition becomes True, we make the bool value of “gameplay” variable as False and make the bool value of “died” variable True, and play a dying sound effect.

```
if player.is_collision():
            screen.gameplay = False
            screen.main_menu = False
            screen.died_music.play()
            screen.died = True
```

- ## COLOUR CHANGING OPTIONS

For the making of colour changing function of the player rect, as well as the colour of the background, we first had to make a list i which there were different RBG values for the colours named colours_rgb. If we do not want to spend our own precious time making different values for the different colours, we could just ask LLMs like ChatGPT or Bard to make us a list consisting of different colours' RGB values. To add different little colour buttons in the Options screen, we go in the OptionsScreen method of the Screens() Class and first blit and render a text on the screen by using pygame.blit() method aswell as text_render() method. then we create a for-loop with enumerate() function in colours_rgb list and we will use the method of our Krekhead() Class named create_button() and increase the x-value by 25, (that is the size of our tiny buttons) so the buttons get blitted, ahead of each other, and for its y-value, we apply a condition that if 10 buttons have been blitted in a single line then we increase the y-value, and then we make the action as appending the pressed buttons RGB value into our player.color list defined in the Krek() Class, and make player rect's colour the value of player.color[-1] (as the append method adds the value at the end of the list). And for the window background colour, we repeat the same process except, instead of the appending the colour value to player.color list, we append the value to krek.theme list.

```
        win.blit(krek.text_render("Assets/Stella.otf", 50,"Choose your colour",
(255,255,255)), (450+120,65))
        for pos,i in enumerate(self.colors_rgb):
            krek.create_button(450+120+(pos*25) if pos <= 10 else (450+(-155)+(pos*25) if
pos    > 10 and pos <=20 else (450+(-405)+(pos*25) if pos > 20 and pos <=30 else (1+2))),
120 if pos <= 10 else (145 if pos > 10 and pos <= 20 else (170 if pos > 20 and pos <= 30
else(5000))), 25, 25, i ,"Assets/Stella.otf" , (255,255,255),"",(0,0,0) ,lambda:
player.color.append(i))
        win.blit(krek.text_render("Assets/Stella.otf", 50,"Choose the colour theme of the
game", (255,255,255)), (450,250))
        for pos,i in enumerate(self.colors_rgb):
            krek.create_button(450+120+(pos*25) if pos <= 10 else (450+(-155)+(pos*25) if
pos > 10 and pos <=20 else (450+(-405)+(pos*25) if pos > 20 and pos <=30 else (1+2))), 310
if pos <= 10 else (335 if pos > 10 and pos <= 20 else (360 if pos > 20 and pos <= 30
else(5000))), 25, 25, i ,"Assets/Stella.otf" , (255,255,255),"",(0,0,0) ,lambda:
krek.theme.append(i))
```

to change the colour of the player, we do;

```
def draw(self, win):
        rect = self.create_character()
        pygame.draw.rect(win, self.color[-1], rect)
```

to change the colour of the screen background, we do;

```
win.fill(krek.theme[-1])
```

5. Conclusion

In conclusion, "Krekhed" delivers an immersive and dynamic gaming experience, featuring intuitive controls, challenging gameplay, and customizable options. Players can enjoy navigating the infinite scrolling background, avoiding obstacles, and striving for high scores. The "You Died" screen provides feedback on performance, allowing players to restart, return to the main menu, or quit the game.

The Options Screen enhances personalization, enabling users to choose the player rectangle color and customize the overall theme color of the program, adding a unique touch to each player's experience. "Krekhed" aims to provide not only entertainment but also a sense of individuality in the gaming world.

With straightforward controls, clear visuals, and responsive gameplay, "Krekhed" invites players of all levels to enjoy a captivating and enjoyable gaming environment. Whether aiming for the high score or exploring customization options, players are encouraged to immerse themselves in the world of "Krekhed" and embark on an exciting gaming journey.