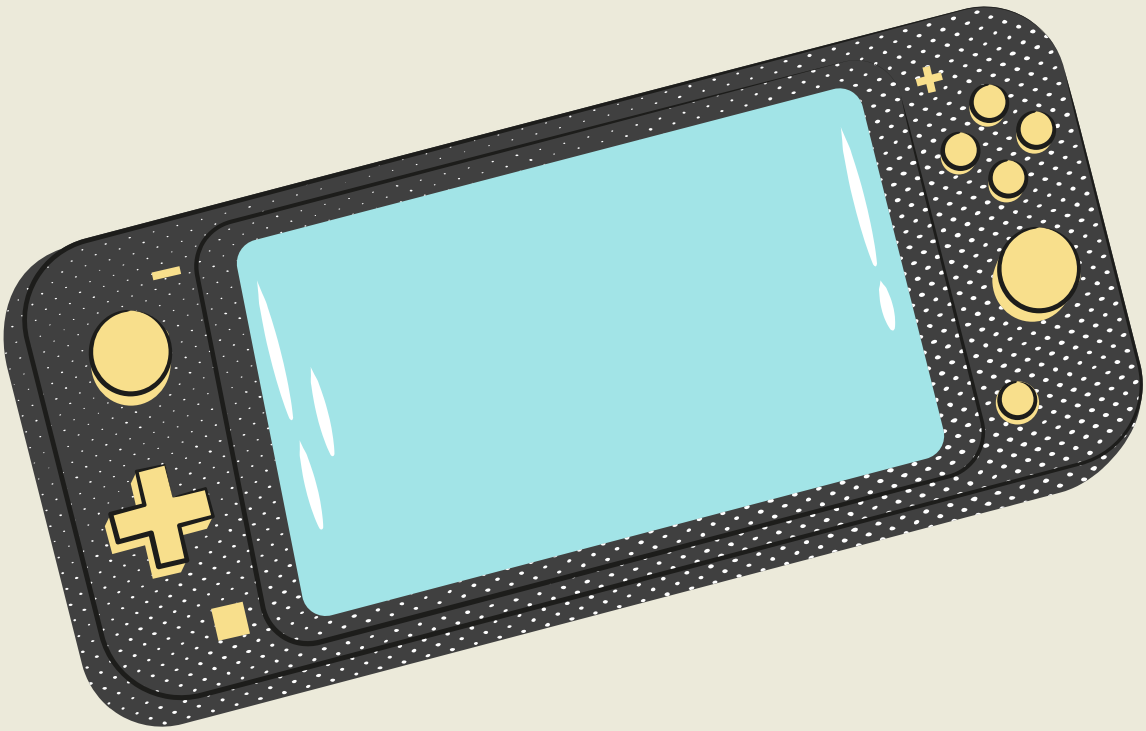
GO UP !



AY 2023 - 24

CBSE COMPUTER PROJECT FILE

DONE BY :

AAFREEN
SUKHMANI
MANTASHA

مدرسة جيه اس اس الخاصة
JSS PRIVATE SCHOOL

CERTIFICATE

This is to certify that Aafreen Niha A of class **XII A** has successfully completed the **Computer Science (083)** Project on the topic Go Up!! during the academic year 2023-2024 as per the guidelines issued by the **Central Board of Secondary Education (New Delhi – India)**

....

Internal Examiner **Signature of External Examiner**

Ms. Rajeswari V
HOD- Computer Science & IT
JSS Private School, Dubai.

مدرسة جيه اس اس الخاصة
JSS PRIVATE SCHOOL

Acknowledgement

I would like to take this opportunity to earnestly thank my Computer Science teacher Mrs. Rajeswari Venkataraman - HOD Computer Science for imparting knowledge in me and providing me with a noteworthy opportunity and her consistent motivation and suitable guidance to work on this exciting project. Last but not the least, I would like to thank my parents for their nurturing support. I had an exceptional experience while coding and the useful insights from the teacher surely did help shape my ideas.

Additionally, I would also like to thank my Grade 11 Computer Science Teacher, Ms. Mega Manjit who guided us with the fundamentals of Python Programming & Supervisor, Ms. Akansha for always believing in us.

Lastly, I would like to put out my sincere gratitude to all my friends who supported me and helped throughout this whole process without which this project would not have been this successful.

TABLE OF CONTENTS

OVERVIEW

The "Go Up" game is designed to be user-friendly and entertaining, suitable for individuals of all ages. The main character, named Jumpy, starts at the bottom of the screen and continuously jumps from one platform to another, working their way upward. The game is easy to understand and navigate.

As Jumpy progresses through the platforms, the player's score gradually increases. This score is then saved in a text file and transferred to a "Highscore" table within a "comp" database. This feature adds a competitive element to the game, encouraging players to strive for higher scores and compete with others.

To add more excitement and challenge to the game, obstacles are introduced as the player's score goes up. These obstacles can include moving platforms and flying enemies. The increasing difficulty level keeps the gameplay engaging and ensures that players remain captivated as they strive to overcome new challenges.

In summary, "Go Up" is a simple yet engaging game where players control Jumpy to ascend through platforms, scoring points along the way. The incorporation of a highscore system and progressively challenging obstacles adds depth and enjoyment to the gaming experience.

OBJECTIVE

The primary objective of the "Go Up" game is to provide an enjoyable and entertaining experience for players. The game is designed with the fundamental purpose of offering fun and amusement to anyone who plays it. The emphasis is on creating an experience that is not only entertaining but also accessible to a broad audience.

Beyond its recreational nature, there is a secondary objective related to educational use. The game can serve as a valuable tool in educational settings, particularly in schools for primary school students. By engaging with the game, young children can benefit from the enhancement of their thinking skills and coordination. The interactive and dynamic nature of the game encourages cognitive development as players navigate through the platforms, make decisions, and respond to challenges.

HARDWARE REQUIREMENT

All you need is a device, such as a laptop or computer. The game does not require an internet connection to play.

SOFTWARE REQUIREMENT

The software requirements include:

- Python
- MySQL
- Python libraries
- IDLE or any text editors

OVERVIEW

The project utilizes a combination of Python and MySQL to implement its functionality. The Python programming language is employed for game development and various tasks related to the operating system, while MySQL serves as the relational database management system to store and manage high scores.

Database Integration: The project incorporates a SQL database, specifically MySQL, to store and retrieve high scores. The data is organized in a table named "**highscore**" within the "**comp**" database. When a player achieves a new high score, the corresponding value is stored in this table. This allows users to access and view all the previous high scores, providing a historical record of gaming achievements.

MODULES

1. **pygame:**
   - Purpose: Used for creating the game itself and importing various features.
   - Role: Enables the development of the game interface and functionality, facilitating the integration of multimedia elements and user interactions.
2. **random**:
   - Purpose: Employed for adding platforms at random locations within the game and adjusting their speed and direction dynamically.
   - Role: Introduces an element of randomness to the game, making each playthrough unique by varying the placement and behavior of platforms.
3. **os**:
   - Purpose: Interacts with the operating system to perform tasks such as creating files and directories, managing files and directories, handling input and output operations, managing environment variables, and overseeing process management.
   - Role: Facilitates the organization and manipulation of files and directories, essential for tasks such as storing game-related data and managing the game environment.
4. **mysql.connector**:
   - Purpose: Establishes connectivity between the Python application and the MySQL database, allowing for the execution of SQL queries and data retrieval.
   - Role: Enables the storage and retrieval of high scores by establishing a connection to the MySQL database, ensuring seamless communication between the game and the database.

FUNCTIONS USED

draw_text :

- Parameters:
  - text: The text to be displayed.
  - font: The font used for the text.
  - text_col: The color of the text.
  - x: The x-coordinate where the text will be drawn.
  - y: The y-coordinate where the text will be drawn.
- Description: This function renders the specified text using the provided font and color and then blits it onto the screen at the specified coordinates.

draw_panel:

- Description: This function draws the information panel at the top of the game window. It includes a colored rectangle, a white line, and the current score.

draw_bg:

- Parameters:
  - bg_scroll: The scrolling value for the background.
- Description: Draws the background image on the screen, creating a scrolling effect by adjusting the y-coordinate based on the bg_scroll parameter.

FUNCTIONS USED

player class:

- Methods:
  - __init__:
    - Parameters:
      - x: Initial x-coordinate of the player.
      - y: Initial y-coordinate of the player.
    - Description: Initializes the player object with attributes such as image, width, height, rectangle, velocity, and flip status.
  - move:
    - Description: Handles player movement based on keyboard input, gravity, and collision with platforms. Updates the player's position and returns the scroll value.
  - draw:
    - Description: Draws the player on the screen, considering the flipping status.

FUNCTIONS USED

Platform class:

- Methods:
  - __init__:
    - Parameters:
      - x: Initial x-coordinate of the platform.
      - y: Initial y-coordinate of the platform.
      - width: Width of the platform.
      - moving: Boolean indicating if the platform is moving.
    - Description: Initializes the platform object with attributes such as image, moving status, move counter, direction, speed, and rectangle.
  - update:
    - Parameters:
      - scroll: The scrolling value for the platforms.
    - Description: Updates the platform's position, making it move side to side if it is a moving platform. Handles changes in direction, updates the vertical position, and checks if the platform has gone off the screen

FUNCTIONS USED

Player Movement and Background Scrolling:

- The mario.move() function is called to handle player movement. The returned scroll value is used to update the background scrolling (bg_scroll).
- The background is scrolled based on the calculated scroll value using the draw_bg function.

Platform and Enemy Updates:

- New platforms and enemies are created based on certain conditions (such as reaching a specific score).
- The update methods of the platform_group and enemy_group are called to update the positions of platforms and enemies, respectively.

Score Update and High Score Display:

- The score is updated based on the positive scroll value.
- A line is drawn at the previous high score, and the current high score is displayed on the screen using the draw_line and draw_text functions.

FUNCTIONS USED

Drawing Game Elements:

- Platforms, enemies, the main character (mario), and the information panel are drawn on the screen

Game Over Handling:

- If the player falls off the screen or collides with an enemy, the game enters the game-over state.
- The death_fx sound is played, and the game over screen is displayed.

Event Handling:

- The event loop checks for quit events (closing the window). If a quit event is detected, the game loop exits.

Display Update:

- The display is updated to reflect the changes made during the current iteration of the game loop.

FUNCTIONS USED

Enemy class (from enemy import Enemy)

- __init__:
  - Parameters:
    - x: Initial x-coordinate of the enemy.
    - y: Initial y-coordinate of the enemy.
    - spritesheet: Spritesheet object for the enemy's animation.
    - speed: Speed of the enemy.
  - Description: Initializes the enemy object with attributes such as initial position, spritesheet, animation frame, and speed.

- update:
  - Parameters:
    - scroll: The scrolling value for the enemies.
    - screen_width: Width of the game screen.
  - Description: Updates the enemy's position and animation frame based on scrolling. Handles changes in the direction and checks if the enemy has gone off the screen.

SOURCE CODE

```
        self.rect.y = y

    def update(self,scroll):
        #moving platfor side to side if it is a moving platform
        if self.moving == True:
            self.move_counter +=1
            self.rect.x += self.direction * self.speed

        #change platform direction if it has move fully or hit a wall
        if self.move_counter >=100 or self.rect.left < 0 or self.rect.right > screen_width:
            self.direction*= -1
            self.move_counter = 0
        #update platforms vertical position
        self.rect.y+=scroll

        #check if platform has gone off the screen:

        if self.rect.top > screen_heigth:
            self.kill()


# player instance
mario = player(screen_width // 2, screen_heigth - 150)

# create sprite groups
platform_group = pygame.sprite.Group()
enemy_group = pygame.sprite.Group()

#create temporary platforms
'''for p in range (MAX_PLATFORMS):
    p_w = random.randint(40, 60)
    p_x = random.randint(0, screen_width - p_w)
    p_y = p * random.randint(80,120)
    platform = Platform(p_x, p_y, p_w)
    platform_group.add(platform)'''
#creating a starting platform manually:
platform=Platform(screen_width//2 - 50,screen_heigth-50,100 , False)
platform_group.add(platform)

# main game loop
run = True
while run:

    clock.tick(FPS)
```

SOURCE CODE

```
    if game_over==False:
        scroll=mario.move()

        #print(scroll)

        # draw bg
        bg_scroll+=scroll
        if bg_scroll>=600:
            bg_scroll=0
        draw_bg(bg_scroll)

        #draw temporary scroll threshold
        #pygame.draw.line(screen,WHITE,(0,SCROLL_THRESH),(screen_width,SCROLL_THRESH))


        # platforms:
        if len(platform_group)< MAX_PLATFORMS:
            p_w=random.randint(40,60)
            p_x=random.randint(0,screen_width - p_w)
            p_y=platform.rect.y - random.randint(80,120)
            p_type = random.randint(1,2)
            if p_type == 1 and score > 500:
                p_moving = True
            else:
                p_moving = False
            platform=Platform(p_x,p_y,p_w,p_moving)
            platform_group.add(platform)




        #update platforms
        platform_group.update(scroll)

        # enemies
        if len(enemy_group) == 0 and score > 1500:
            enemy = Enemy(screen_width, 100, bird_sheet, 1.5)
            enemy_group.add(enemy)

        #update enemies
        enemy_group.update(scroll, screen_width)


        #update score
        if scroll > 0:
```

SOURCE CODE

```
        #draw line at previous highscore
        pygame.draw.line(screen, white,(0, score - high_score + SCROLL_THRESH),(screen_width, score - high_score + SCROLL_THRESH),3)
        draw_text('HIGH SCORE', font_small, BLACK, screen_width - 130, score - high_score + SCROLL_THRESH)

        # draw main character
        platform_group.draw(screen)
        enemy_group.draw(screen)
        mario.draw()

        #draw panel
        draw_panel()

        #check if the game is over:
        if mario.rect.top > screen_heigth:
            game_over=True
            death_fx.play()

        #check for collison with enemies
        if pygame.sprite.spritecollide(mario, enemy_group, False):
            if pygame.sprite.spritecollide(mario,enemy_group, False, pygame.sprite.collide_mask):
                game_over = True
                death_fx.play()
    else:
        if fade_counter<screen_width:
            fade_counter+=5
            for y in range(0,6,2):
                pygame.draw.rect(screen,BLACK,(0,y*100,fade_counter,100))
                pygame.draw.rect(screen,BLACK,(screen_width - fade_counter,(y+1)*100,screen_width,100))
        else:
            draw_text('GAME OVER!!',font_big,white,130,200)
            draw_text('SCORE:' + str(score),font_big,white,130,250)
            draw_text('PRESS SPACE TO PLAY AGAIN!',font_big,white,40,300)
        #update high score
        if score > high_score:
            high_score = score
            with open('score.txt','w') as file:
                file.write(str(high_score))
        key=pygame.key.get_pressed()
        if key[pygame.K_SPACE]:
            #reset variables:
            game_over=False
            score=0
            scroll=0
            fade_counter=0
            #reposition player
```

SOURCE CODE

```
            fade_counter=0
            #reposition player
            mario.rect.center =(screen_width // 2, screen_heigth - 150)

            #reset enemies
            enemy_group.empty()


            #reset platforms
            platform_group.empty()

            #creating a starting platform manually:
            platform=Platform(screen_width//2 - 50,screen_heigth-50,100, False)
            platform_group.add(platform)

    # event handler
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            #update high score
            if score > high_score:
                high_score = score
                with open('score.txt','w') as file:
                    file.write(str(high_score))
            run = False

    # update display window
    pygame.display.update()


pygame.quit()
```

SOURCE CODE

```python
import pygame

class Spritesheet():
    def __init__(self, image):
        self.sheet = image

    def get_image(self, frame, width, height, scale, colour):
        image = pygame.Surface((width, height)).convert_alpha()  # Use SRCALPHA for transparency
        image.blit(self.sheet, (0, 0), ((frame * width), 0, width, height))
        image = pygame.transform.scale(image, (int(width * scale), int(height * scale)))  # Close the parenthesis
        image.set_colorkey(colour)

        return image

```

```python
import pygame
import random

class Enemy(pygame.sprite.Sprite):
    def __init__(self, screen_width, y, sprite_sheet, scale):
        pygame.sprite.Sprite.__init__(self)
        #define variables
        self.animation_list = []
        self.frame_index = 0
        self.update_time = pygame.time.get_ticks()

        self.direction = random.choice([-1,1])
        if self.direction == 1:
            self.flip = True
        else:
            self.flip = False



        #load images from spritesheet
        animation_steps = 4
        for animation in range(animation_steps):
            image = sprite_sheet.get_image(animation,32,32,scale,(0,0,0))
            image = pygame.transform.flip(image,self.flip, False)
            image.set_colorkey((0,0,0))
            self.animation_list.append(image)

        #select starting image and create a rectangle from it
        self.image = self.animation_list[self.frame_index]
        self.rect = self.image.get_rect()

        if self.direction == 1:
            self.rect.x =0

        else:
            self.rect.x = screen_width

        #self.rect.x = 0
        self.rect.y = y

    def update(self,scroll, screen_width):
```

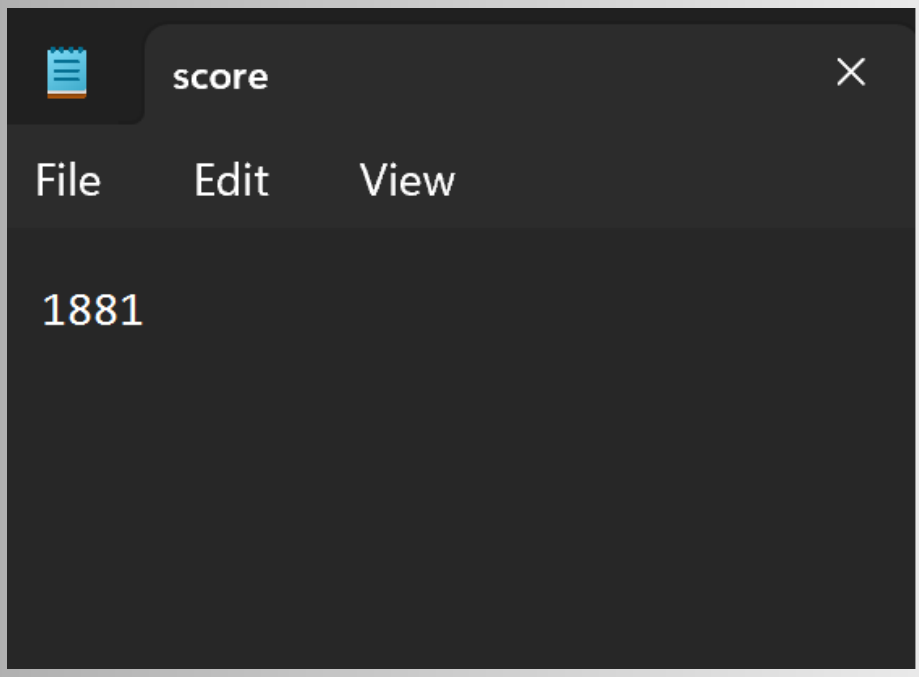
SOURCE CODE

```
    def update(self,scroll, screen_width):

        #update animation
        animation_cooldown = 50
        #update image depending on current frame
        self.image = self.animation_list[self.frame_index]
        #check if enough time has passed since the last update
        if pygame.time.get_ticks() - self.update_time > animation_cooldown:
            self.update_time = pygame.time.get_ticks()
            self.frame_index += 1
        #if the animation has run out reset back to the start
        if self.frame_index >= len(self.animation_list):
            self.frame_index = 0




        #move enemy
        self.rect.x += self.direction * 2
        self.rect.y += scroll

        #check if gone off screen
        if self.rect.right < 0 or self.rect.left > screen_width:
            self.kill()

```

OUTPUT

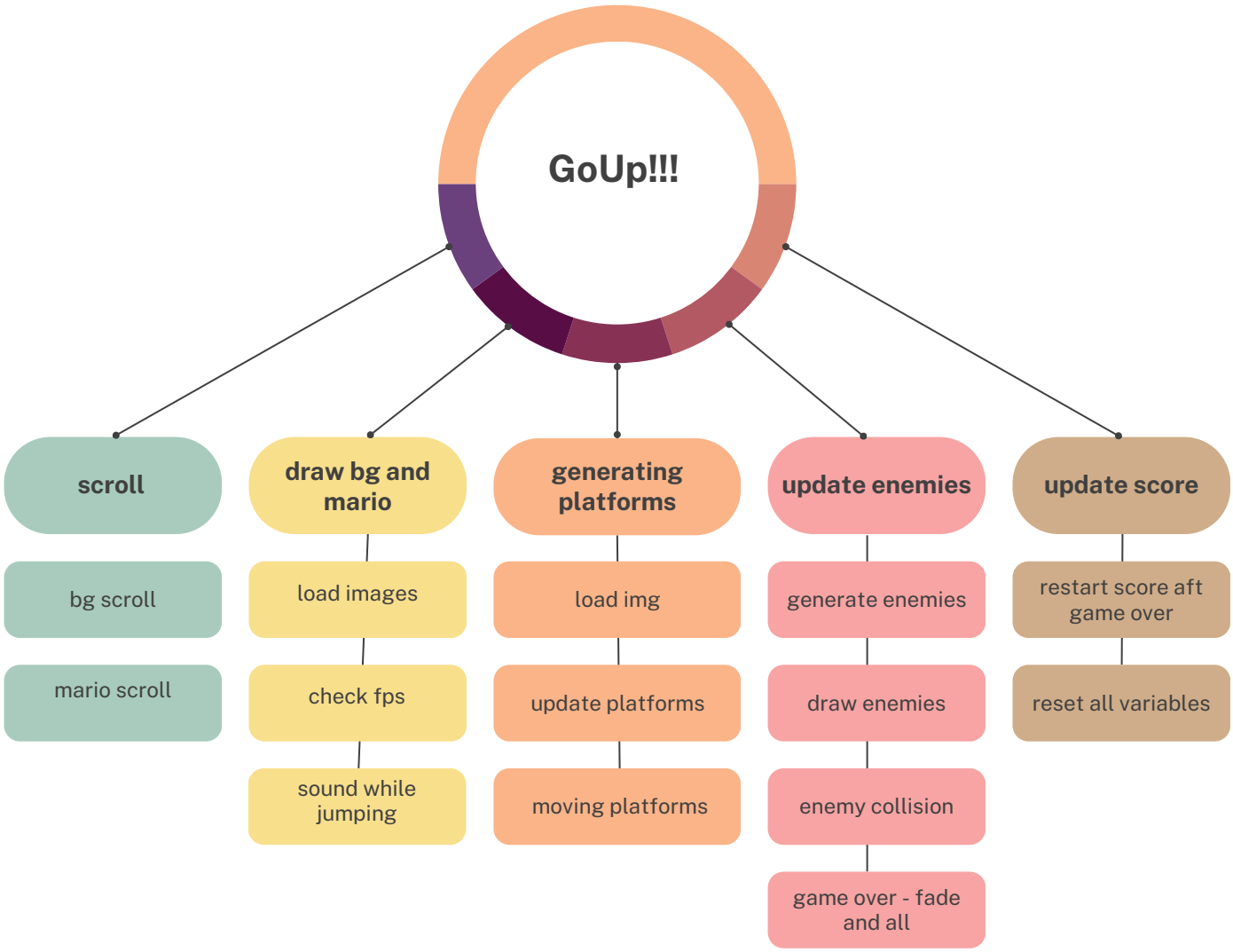
OUTPUT

HIGH SCORE STORED IN BOTH TEXT FILE AND IN SQL




```
mysql> use proj;
Database changed
mysql> show tables;
+----------------+
| Tables_in_proj |
+----------------+
| highscore      |
+----------------+
1 row in set (0.03 sec)

mysql> select * from highscore;
+-------+
| score |
+-------+
|  1881 |
+-------+
1 row in set (0.00 sec)
```

FLOWCHART


GoUp!!!
scroll
bg scroll
mario scroll
draw bg and mario
load images
check fps
sound while jumping
generating platforms
load img
update platforms
moving platforms
update enemies
generate enemies
draw enemies
enemy collision
game over - fade and all
update score
restart score aft game over
reset all variables