Project Report

Platformers Game

Group ID :- 43
Shikha Chauhan – 4163
Kanvi Shah – 4215

Supervisor: Hitesh Parmar

Department Name : Msc(CA & IT)

College Name : K.S. school of business management

University Name : Gujarat University

TABLE OF CONTENT

Abstract 3

Chapter 1 : Introduction 4

1.1 How to use this template 4

Chapter 2 : Project 5

2.1 Source code 6

2.2 Result 17

Chapter 3 : Conclusion 21

Reference 22

Abstract

This project is a platformer game. In this game there is a player which will be controlled by the user manually. In this game the player needs to save his life from the **potential enemy** provided in this game along with that the player needs to jump appropriately to reach to the next block and not fell down as it results in loosing the game. As this game requires jumping as essential part we have used **bouncing algorithm** to fulfil this requirement.

CHAPTER 1: INTRODUCTION

Importance Of This Topic:

- Our project is about developing a platformer game using bouncing algorithm. Soul purpose behind creating this game is to help player in gaining entertainment by playing this game.

- Nowadays people prefer playing games on laptop or desktop rather than television video games or mobile phone games so this game will gain popularity in the coming future.

Use Of The Application:

- This project is useful in developing the laptop or desktop based games.

- This game is an upgraded version of the old popular television game called super Mario so this will be useful in the game development and can be used in the video games too.

Important Talk About This Project:

- This game is basically an updated version of old television video game.
- In this game there is a player which will be manually operated by the player.
- This is a laptop or desktop game.
- In this game we have used bouncing algorithm to develop this project.
- The user will reach to the next level after completing the level.
- The user also needs to collect the coins which appear in the way to increase the overall score of the player.
- In this game we have created some enemies from which the player needs to save his life , touching that enemies cause game over.
- Along with that the user needs to jump properly to reach to the next block or else he will fell down which also results in the end of the game.

CHAPTER 2: PROJECT

Approach To The Problem:

- In this platformer game the basic idea is that the player needs to jump to reach to that block
- In order to resolve this problem we have used a well-known algorithm called bouncing algorithm
- In order to display the game properly on the screen we have used grid functionality.
- In this game blocks are moving, to achieve this we have used float property.

DESIGN:

What We Already Had:

- In making of this game we had photographs related to our game.
- Pictures related to the main player of the game, game background, blocks, enemies were taken from git hub.
- We also understood the code of the bouncing algorithm and how the grid property works in python.

Added Things To The Project:

- In this project we have created 7 different levels till now.
- In order to make this game more interesting we have added the sound effects to this game.
- To catch the user attention we have created different animation at the time of the game over.
- To make this game more user friendly we have added 7 different screen for each level.
- The feature of coin collection is also added to the game.

2.1 Source code:

```
import pygame
from pygame.locals import *
from pygame import mixer
import pickle
from os import path

pygame.mixer.pre_init(44100, -16, 2, 512)
mixer.init()
pygame.init()

clock = pygame.time.Clock()
fps = 60

screen_width = 680
screen_height = 680

screen = pygame.display.set_mode((screen_width, screen_height))
pygame.display.set_caption('Platformer')



#define font
font = pygame.font.SysFont('Bauhaus 93', 70)
font_score = pygame.font.SysFont('Bauhaus 93', 30)



#define game variables
tile_size = 34
game_over = 0
main_menu = True
level = 1
max_levels = 7
score = 0
```

```
#define colours
white = (255, 255, 255)
blue = (0, 0, 255)


#load images
sun_img = pygame.image.load('img/sun.png')
bg_img = pygame.image.load('img/sky.png')
restart_img = pygame.image.load('img/restart_btn.png')
start_img = pygame.image.load('img/start_btn.png')
exit_img = pygame.image.load('img/exit_btn.png')

#load sounds
pygame.mixer.music.load('img/music.wav')
pygame.mixer.music.play(-1, 0.0, 5000)
coin_fx = pygame.mixer.Sound('img/coin.wav')
coin_fx.set_volume(0.5)
jump_fx = pygame.mixer.Sound('img/jump.wav')
jump_fx.set_volume(0.5)
game_over_fx = pygame.mixer.Sound('img/game_over.wav')
game_over_fx.set_volume(0.5)


def draw_text(text, font, text_col, x, y):
      img = font.render(text, True, text_col)
      screen.blit(img, (x, y))


#function to reset level
def reset_level(level):
      player.reset(100, screen_height - 130)
      blob_group.empty()
      platform_group.empty()
      coin_group.empty()
```

```
	lava_group.empty()
	exit_group.empty()

	#load in level data and create world
	if path.exists(f'level{level}_data'):
		pickle_in = open(f'level{level}_data', 'rb')
		world_data = pickle.load(pickle_in)
	world = World(world_data)
	#create dummy coin for showing the score
	score_coin = Coin(tile_size // 2, tile_size // 2)
	coin_group.add(score_coin)
	return world


class Button():
	def __init__(self, x, y, image):
		self.image = image
		self.rect = self.image.get_rect()
		self.rect.x = x
		self.rect.y = y
		self.clicked = False

	def draw(self):
		action = False

		#get mouse position
		pos = pygame.mouse.get_pos()

		#check mouseover and clicked conditions
		if self.rect.collidepoint(pos):
			if pygame.mouse.get_pressed()[0] == 1 and self.clicked ==
False:
				action = True
				self.clicked = True

		if pygame.mouse.get_pressed()[0] == 0:
```

```
                self.clicked = False



            #draw button
            screen.blit(self.image, self.rect)

            return action



class Player():
      def __init__(self, x, y):
            self.reset(x, y)

      def update(self, game_over):
            dx = 0
            dy = 0
            walk_cooldown = 5
            col_thresh = 20

            if game_over == 0:
                  #get keypresses
                  key = pygame.key.get_pressed()
                  if key[pygame.K_SPACE] and self.jumped == False and
self.in_air == False:
                        jump_fx.play()
                        self.vel_y = -15
                        self.jumped = True
                  if key[pygame.K_SPACE] == False:
                        self.jumped = False
                  if key[pygame.K_LEFT]:
                        dx -= 5
                        self.counter += 1
                        self.direction = -1
                  if key[pygame.K_RIGHT]:
                        dx += 5
                        self.counter += 1
```

```
                        self.direction = 1
                if key[pygame.K_LEFT] == False and key[pygame.K_RIGHT] ==
False:
                        self.counter = 0
                        self.index = 0
                        if self.direction == 1:
                                self.image = self.images_right[self.index]
                        if self.direction == -1:
                                self.image = self.images_left[self.index]



                #handle animation
                if self.counter > walk_cooldown:
                        self.counter = 0
                        self.index += 1
                        if self.index >= len(self.images_right):
                                self.index = 0
                        if self.direction == 1:
                                self.image = self.images_right[self.index]
                        if self.direction == -1:
                                self.image = self.images_left[self.index]



                #add gravity
                self.vel_y += 1
                if self.vel_y > 10:
                        self.vel_y = 10
                dy += self.vel_y


                #check for collision
                self.in_air = True
                for tile in world.tile_list:
                        #check for collision in x direction
                        if tile[1].colliderect(self.rect.x + dx,
self.rect.y, self.width, self.height):
                                dx = 0
                        #check for collision in y direction
```

```
                        if tile[1].colliderect(self.rect.x, self.rect.y +
dy, self.width, self.height):
                                #check if below the ground i.e. jumping
                                if self.vel_y < 0:
                                        dy = tile[1].bottom - self.rect.top
                                        self.vel_y = 0
                                #check if above the ground i.e. falling
                                elif self.vel_y >= 0:
                                        dy = tile[1].top - self.rect.bottom
                                        self.vel_y = 0
                                        self.in_air = False



                #check for collision with enemies
                if pygame.sprite.spritecollide(self, blob_group, False):
                        game_over = -1
                        game_over_fx.play()

                #check for collision with lava
                if pygame.sprite.spritecollide(self, lava_group, False):
                        game_over = -1
                        game_over_fx.play()

                #check for collision with exit
                if pygame.sprite.spritecollide(self, exit_group, False):
                        game_over = 1



                #check for collision with platforms
                for platform in platform_group:
                        #collision in the x direction
                        if platform.rect.colliderect(self.rect.x + dx,
self.rect.y, self.width, self.height):
                                dx = 0
                        #collision in the y direction
                        if platform.rect.colliderect(self.rect.x,
self.rect.y + dy, self.width, self.height):
```

```
                                    #check if below platform
                                    if abs((self.rect.top + dy) -
platform.rect.bottom) < col_thresh:
                                          self.vel_y = 0
                                          dy = platform.rect.bottom -
self.rect.top
                                    #check if above platform
                                    elif abs((self.rect.bottom + dy) -
platform.rect.top) < col_thresh:
                                          self.rect.bottom = platform.rect.top -
1
                                          self.in_air = False
                                          dy = 0
                                    #move sideways with the platform
                                    if platform.move_x != 0:
                                          self.rect.x += platform.move_direction



                        #update player coordinates
                        self.rect.x += dx
                        self.rect.y += dy



                elif game_over == -1:
                        self.image = self.dead_image
                        draw_text('GAME OVER!', font, blue, (screen_width // 2) -
200, screen_height // 2)
                        if self.rect.y > 200:
                              self.rect.y -= 5


                #draw player onto screen
                screen.blit(self.image, self.rect)


                return game_over



        def reset(self, x, y):
                self.images_right = []
```

```
            self.images_left = []
            self.index = 0
            self.counter = 0
            for num in range(1, 5):
                  img_right = pygame.image.load(f'img/guy{num}.png')
                  img_right = pygame.transform.scale(img_right, (40, 80))
                  img_left = pygame.transform.flip(img_right, True, False)
                  self.images_right.append(img_right)
                  self.images_left.append(img_left)
            self.dead_image = pygame.image.load('img/ghost.png')
            self.image = self.images_right[self.index]
            self.rect = self.image.get_rect()
            self.rect.x = x
            self.rect.y = y
            self.width = self.image.get_width()
            self.height = self.image.get_height()
            self.vel_y = 0
            self.jumped = False
            self.direction = 0
            self.in_air = True



class World():
      def __init__(self, data):
            self.tile_list = []

            #load images
            dirt_img = pygame.image.load('img/dirt.png')
            grass_img = pygame.image.load('img/grass.png')

            row_count = 0
            for row in data:
                  col_count = 0
                  for tile in row:
                        if tile == 1:
```

```
                                img = pygame.transform.scale(dirt_img,
(tile_size, tile_size))
                                img_rect = img.get_rect()
                                img_rect.x = col_count * tile_size
                                img_rect.y = row_count * tile_size
                                tile = (img, img_rect)
                                self.tile_list.append(tile)
                          if tile == 2:
                                img = pygame.transform.scale(grass_img,
(tile_size, tile_size))
                                img_rect = img.get_rect()
                                img_rect.x = col_count * tile_size
                                img_rect.y = row_count * tile_size
                                tile = (img, img_rect)
                                self.tile_list.append(tile)
                          if tile == 3:
                                blob = Enemy(col_count * tile_size, row_count
* tile_size + 15)
                                blob_group.add(blob)
                          if tile == 4:
                                platform = Platform(col_count * tile_size,
row_count * tile_size, 1, 0)
                                platform_group.add(platform)
                          if tile == 5:
                                platform = Platform(col_count * tile_size,
row_count * tile_size, 0, 1)
                                platform_group.add(platform)
                          if tile == 6:
                                lava = Lava(col_count * tile_size, row_count
* tile_size + (tile_size // 2))
                                lava_group.add(lava)
                          if tile == 7:
                                coin = Coin(col_count * tile_size +
(tile_size // 2), row_count * tile_size + (tile_size // 2))
                                coin_group.add(coin)
                          if tile == 8:
                                exit = Exit(col_count * tile_size, row_count
* tile_size - (tile_size // 2))
                                exit_group.add(exit)
```

```
                        col_count += 1
                row_count += 1


    def draw(self):
        for tile in self.tile_list:
            screen.blit(tile[0], tile[1])



class Enemy(pygame.sprite.Sprite):
    def __init__(self, x, y):
        pygame.sprite.Sprite.__init__(self)
        self.image = pygame.image.load('img/blob.png')
        self.rect = self.image.get_rect()
        self.rect.x = x
        self.rect.y = y
        self.move_direction = 1
        self.move_counter = 0

    def update(self):
        self.rect.x += self.move_direction
        self.move_counter += 1
        if abs(self.move_counter) > 50:
            self.move_direction *= -1
            self.move_counter *= -1



class Platform(pygame.sprite.Sprite):
    def __init__(self, x, y, move_x, move_y):
        pygame.sprite.Sprite.__init__(self)
        img = pygame.image.load('img/platform.png')
        self.image = pygame.transform.scale(img, (tile_size, tile_size
// 2))
        self.rect = self.image.get_rect()
        self.rect.x = x
```

```
            self.rect.y = y
            self.move_counter = 0
            self.move_direction = 1
            self.move_x = move_x
            self.move_y = move_y


      def update(self):
            self.rect.x += self.move_direction * self.move_x
            self.rect.y += self.move_direction * self.move_y
            self.move_counter += 1
            if abs(self.move_counter) > 50:
                  self.move_direction *= -1
                  self.move_counter *= -1




class Lava(pygame.sprite.Sprite):
      def __init__(self, x, y):
            pygame.sprite.Sprite.__init__(self)
            img = pygame.image.load('img/lava.png')
            self.image = pygame.transform.scale(img, (tile_size, tile_size
// 2))
            self.rect = self.image.get_rect()
            self.rect.x = x
            self.rect.y = y


class Coin(pygame.sprite.Sprite):
      def __init__(self, x, y):
            pygame.sprite.Sprite.__init__(self)
            img = pygame.image.load('img/coin.png')
            self.image = pygame.transform.scale(img, (tile_size // 2,
tile_size // 2))
            self.rect = self.image.get_rect()
```

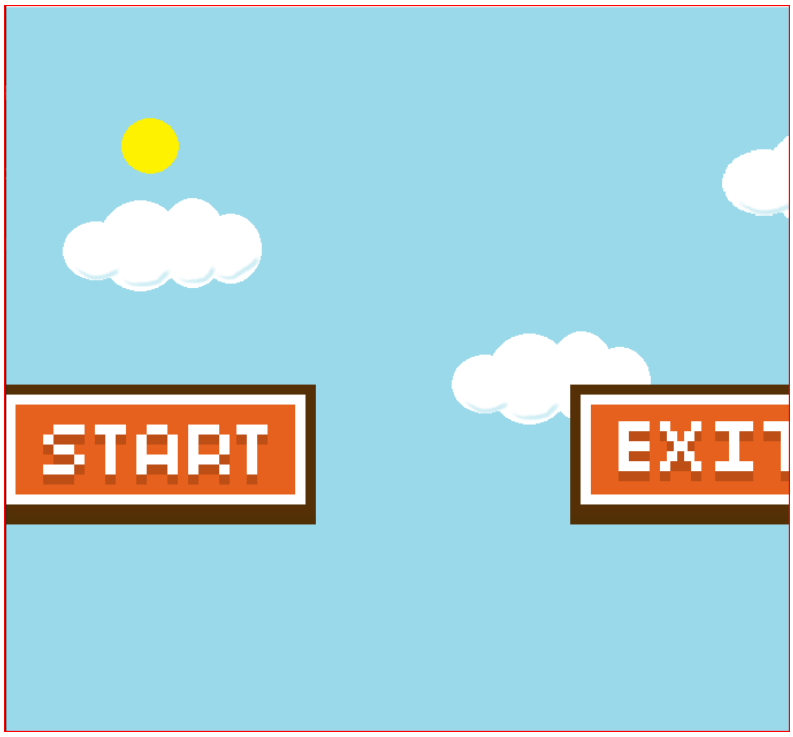
```
            self.rect.center = (x, y)



class Exit(pygame.sprite.Sprite):
      def __init__(self, x, y):
            pygame.sprite.Sprite.__init__(self)
            img = pygame.image.load('img/exit.png')
            self.image = pygame.transform.scale(img, (tile_size,
int(tile_size * 1.5)))
            self.rect = self.image.get_rect()
            self.rect.x = x
            self.rect.y = y




player = Player(100, screen_height - 130)


blob_group = pygame.sprite.Group()
platform_group = pygame.sprite.Group()
lava_group = pygame.sprite.Group()
coin_group = pygame.sprite.Group()
exit_group = pygame.sprite.Group()


#create dummy coin for showing the score
score_coin = Coin(tile_size // 2, tile_size // 2)
coin_group.add(score_coin)


#load in level data and create world
if path.exists(f'level{level}_data'):
      pickle_in = open(f'level{level}_data', 'rb')
      world_data = pickle.load(pickle_in)
world = World(world_data)



#create buttons
restart_button = Button(screen_width // 2 - 50, screen_height // 2 + 100,
restart_img)
```

```
start_button = Button(screen_width // 2 - 350, screen_height // 2,
start_img)
exit_button = Button(screen_width // 2 + 150, screen_height // 2, exit_img)



run = True
while run:

	clock.tick(fps)

	screen.blit(bg_img, (0, 0))
	screen.blit(sun_img, (100, 100))

	if main_menu == True:
		if exit_button.draw():
			run = False
		if start_button.draw():
			main_menu = False
	else:
		world.draw()

		if game_over == 0:
			blob_group.update()
			platform_group.update()
			#update score
			#check if a coin has been collected
			if pygame.sprite.spritecollide(player, coin_group, True):
				score += 1
				coin_fx.play()
			draw_text('X ' + str(score), font_score, white, tile_size
- 10, 10)

		blob_group.draw(screen)
		platform_group.draw(screen)
		lava_group.draw(screen)
		coin_group.draw(screen)
		exit_group.draw(screen)
```

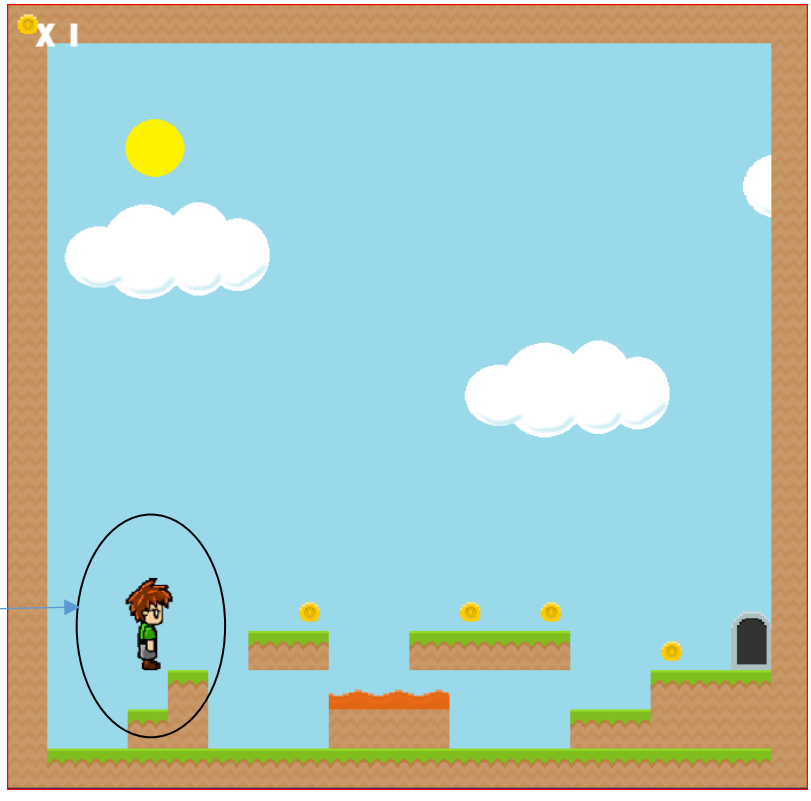
```
            game_over = player.update(game_over)

            #if player has died
            if game_over == -1:
                  if restart_button.draw():
                        world_data = []
                        world = reset_level(level)
                        game_over = 0
                        score = 0

            #if player has completed the level
            if game_over == 1:
                  #reset game and go to next level
                  level += 1
                  if level <= max_levels:
                        #reset level
                        world_data = []
                        world = reset_level(level)
                        game_over = 0
                  else:
                        draw_text('YOU WIN!', font, blue, (screen_width //
2) - 140, screen_height // 2)
                        if restart_button.draw():
                              level = 1
                              #reset level
                              world_data = []
                              world = reset_level(level)
                              game_over = 0
                              score = 0

      for event in pygame.event.get():
            if event.type == pygame.QUIT:
                  run = False

      pygame.display.update()  pygame.quit()
```

2.2 Result:

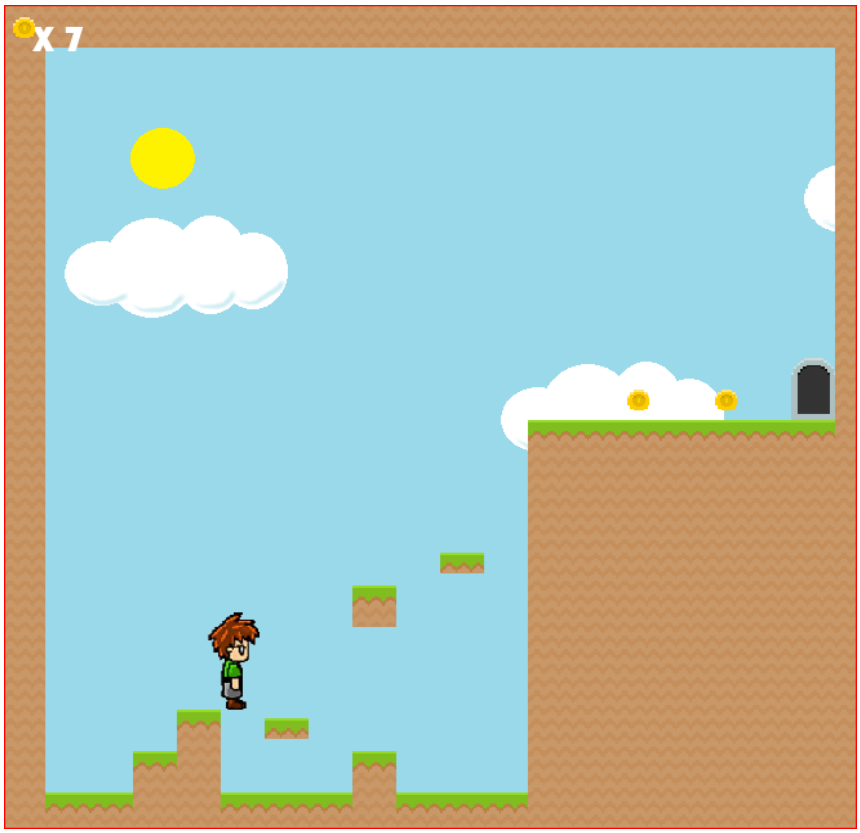
Starting screen



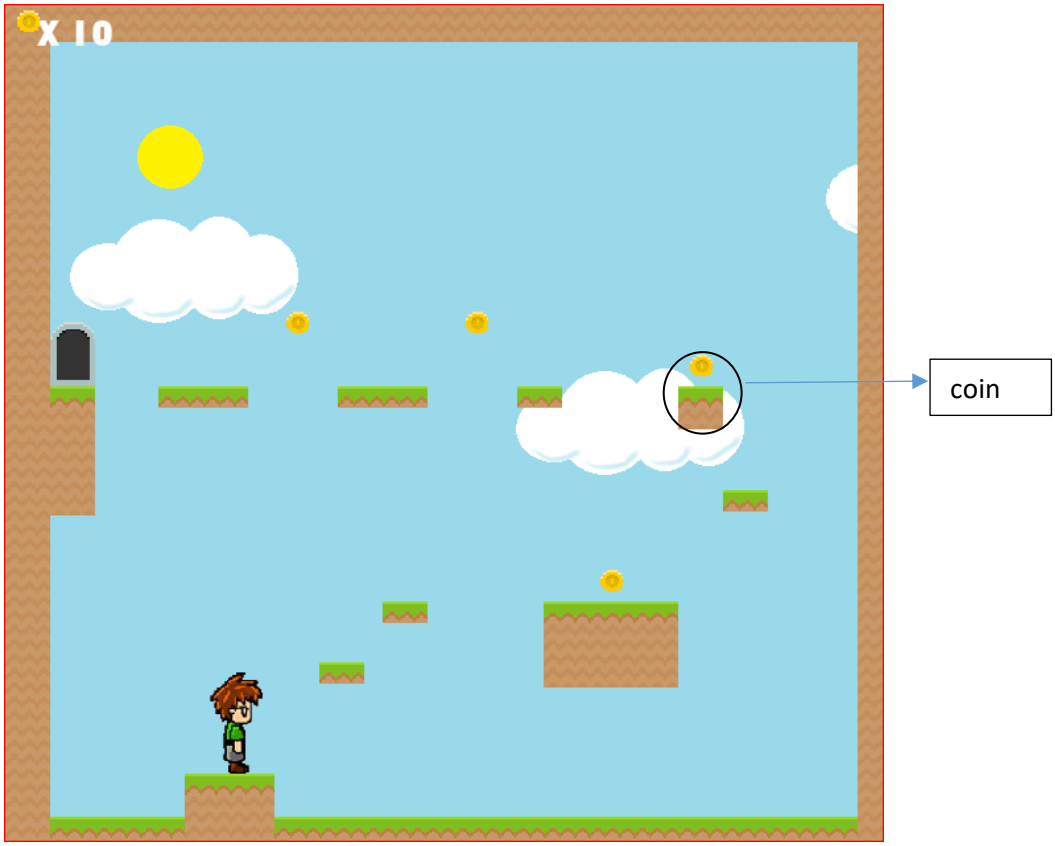
Level 1



Level 2



Level 3



Level 4

Last level

game over

chapter 3: CONCLUSION

Things That Have Been Learned:

- handling images in python games
- concept of grid
- how to use AI algorithm for game development (bouncing algorithm)
- how to create objects and make them work as real entities.
- Adding multiple screens in a single project.
- Adding sound facility
- Usage of Git and GitHub
- Use of inbuilt python libraries according to project requirement.

Different Scenario Of Doing This Project:

- Instead of providing grid to each and every block a new property called grid-gap can be used.
- In this project we could have used image processing for better user experience.
- This project can also use A* algorithm in order to find the shortest path to reach to the destination.

Future Work:

- As of now we have created seven different levels in this game. which we will develop till 100 levels for the players.
- At the current stage the player in unable to kill the enemy by bullets, we are also planning to add this feature in the coming future.

References:

- GitHub Link : https://github.com/russs123/Platformer.git
- YouTube channel Link : https://www.youtube.com/watch?v=Ongc4EVqRjo&t=353s
- Bouncing algorithm video : https://www.youtube.com/watch?v=YIKRXl3wH8Y
- Bouncing algorithm document : https://www.101computing.net/bouncing-algorithm/
- Python reference : https://www.python.org/doc/