PROJECT REPORT

MARCH 25

13657779
Yuvraj Singh Cheema

Certificate

This is to certify that Yuvraj Singh Cheema of XII (Non-Medical) has completed his project file in partial fulfillment of the requirements set by the CBSE. It is authentic work carried out by him under my supervision. He has taken proper care and shown utmost sincerity in completion of the project.

I certify that this project is up to my expectations and as per the guidelines issued by the CBSE.

Ms. Prabhjit Kaur
(PGT Computer Science)

Acknowledgement

I would like to express my special thanks of gratitude to my teacher Ms. Prebhjit Kaur as well as our principal Ms. Manbir Brar who gave me the golden opportunity to do this wonderful project on the topic Python, which also helped me in doing a lot of research and I came to know about many new things I am really thankful to them.

Secondly I would also like to thank my parents and friends who helped me a lot in finalizing this project within the limited time frame

Yuvraj Singh Cheema

Index

Analysis

As I began the project I knew I wanted to make a game that can be used as a good break time game, so I had two options: first was to have a game with small levels and the second to make an endless one. Since this was meant to be used during breaks the latter made more sense as it had more replayability. To make sure the users don't overplay it needed a steep difficulty curve so most average users play for shorter times and do not delay their responsibilities. To have a large userbase the game was designed with simple graphics and minimal controls so that it can be played on most computers.

System Requirements

Hardware

Min:

Intel Pentium Dual Core or AMD equivalent
1.5 GB RAM
1 GB empty storage

Recommended:

Intel Core i3 (5th Gen) or AMD equivalent
4 GB RAM
1 GB empty storage

Software

Min:

Windows 7 (32-bit)
Python 3.7 (with pygame library installed)

Recommended:

Windows 10 (64-bit)
Python 3.7 (with pygame library installed)

Feasibility Study

Feasibility study is a system proposal according to its work, ability, impact on the operation ability to meet the needs of users and efficient use of resources. An important outcome of preliminary investigations the determination of that system requested feasible.

ECONOMICAL FEASIBILITY:

Economics analysis is the most frequent use method for evaluating the effectiveness of the candidates the benefits and savings that are expected from system and compare them with cost.

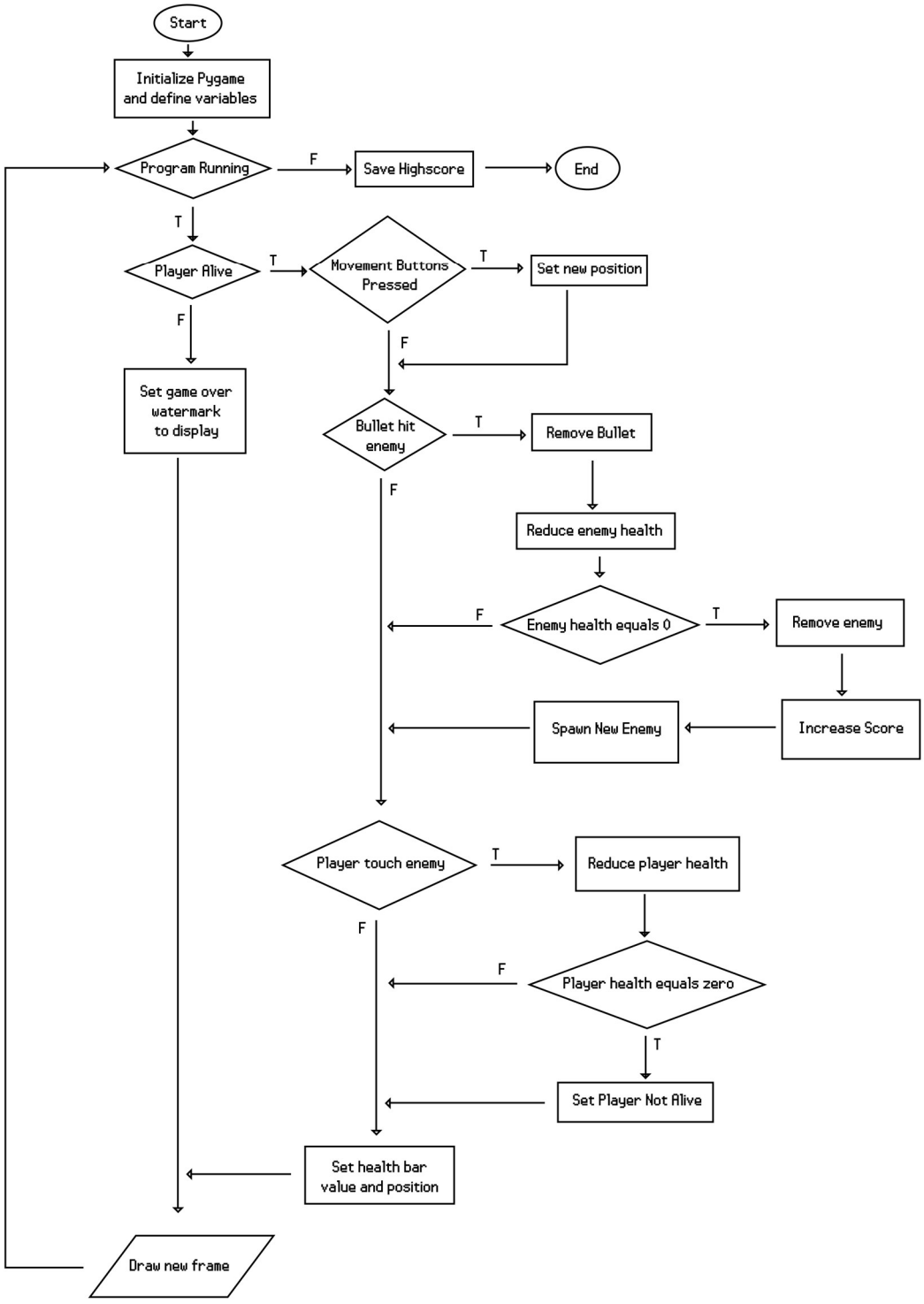
Though most working people don't give much importance to them, video games are a good stress buster which in turn helps everyone be more productive therefore making it economically feasible.

TECHNICAL FEASIBILITY:

Technical feasibility center on the existing computer system and to what extent it can support the proposed task. This involves financial consideration to accommodate technical enhancements.

It is technically feasible because whatever technology is needed to develop this software is easily available.

Flow Chart of the Program



Code

```
import pygame
import time
import random

pygame.init()

window_width = 800  # px
window_height = 450  # px
window = pygame.Surface((800, 450))
main_window = pygame.display.set_mode((1600, 900))
pygame.display.set_caption("Pixelpoclapse")


class sounds():
    enemyHurt = pygame.mixer.Sound('enemyHurt.wav')
    enemyDeath = pygame.mixer.Sound('enemyDeath.wav')
    playerHurt = pygame.mixer.Sound('hurt.wav')
    gunShoot = pygame.mixer.Sound('shoot.wav')
    gunReload = pygame.mixer.Sound('reload.wav')
    reloaded = False

    music = pygame.mixer.music.load('music.mp3')


class player(object):

    def __init__(self, x, y, w, h, health):
        self.height = h
        self.width = w
        self.x_pos = x
        self.y_pos = y
        self.movement_speed = 9
        self.color = 255, 100, 100
        self.isJump = False
        self.jumpCount = 10
        self.direction = 1  # useful for projectile 1 means right -1 means left
        self.max_ammo = 6
        self.curr_ammo = 6
        self.reloading = False
        self.lastReloadTime = None
        self.hurtTime = 0
        self.immune = False
        self.visible = True
        self.maxHealth = health
        self.currHealth = 1 + self.maxHealth - 1
        self.score = 0
        self.alive = True
```

```
    def movement(self):

        jumpKeyPressed = keys[pygame.K_SPACE] or keys[pygame.K_UP] or
keys[pygame.K_w]

        if (keys[pygame.K_LEFT] or keys[pygame.K_a]) and self.x_pos > 0 +
self.movement_speed:
            self.x_pos -= self.movement_speed
            self.direction = -1

        if (keys[pygame.K_RIGHT] or keys[pygame.K_d]) and self.x_pos <
window_width - self.width - self.movement_speed:
            self.x_pos += self.movement_speed
            self.direction = 1

        if not self.isJump:
            if jumpKeyPressed:
                self.isJump = True
        else:
            if self.jumpCount >= -10:
                self.y_pos -= self.jumpCount * abs(self.jumpCount) * 0.69
                self.jumpCount -= 2
            else:
                self.isJump = False
                self.jumpCount = 10

    def draw(self):
        pygame.draw.rect(window, self.color, (self.x_pos, self.y_pos, self.width,
self.height))  # character
        if self.direction > 0:
            pygame.draw.rect(window, (42, 52, 57),
                             (self.x_pos + self.width - 5, round(self.y_pos +
(self.height // 2) - 3), 10, 6))  # gun
            pygame.draw.rect(window, (42, 52, 57),
                             (self.x_pos + self.width - 5 - 4, round(self.y_pos +
(self.height // 2) - 3), 4, 10))  # gun

        else:
            pygame.draw.rect(window, (42, 52, 57),
                             (self.x_pos - 5, round(self.y_pos + (self.height // 2) -
3), 10, 6))  # gun
            pygame.draw.rect(window, (42, 52, 57),
                             (self.x_pos + 5, round(self.y_pos + (self.height // 2) -
3), 4, 10))  # gun
        pygame.draw.rect(window, (255, 0, 0), (self.x_pos, self.y_pos - 10,
self.width, 4))  # health bar red
        pygame.draw.rect(window, (0, 255, 0), (
            self.x_pos, self.y_pos - 10, (self.currHealth / self.maxHealth) *
self.width, 4))  # health bar green
```

```
    def enemy_touch(self):
        sounds.playerHurt.play()
        self.hurtTime = 1
        self.currHealth -= 1


class projectile(object):

    def __init__(self, x, y, direction):
        self.x_pos = x
        self.y_pos = y
        self.radius = 2
        self.vel = 15 * direction
        self.color = 255, 0, 0

    def draw(self):
        pygame.draw.circle(window, self.color, (self.x_pos, self.y_pos),
self.radius)


class enemy(object):

    def __init__(self, x, y, end, width, height, health, aitype):
        self.height = height
        self.width = width
        self.x_pos = x
        self.y_pos = y
        self.vel = 5
        self.color = 84, 197, 113
        self.pathEnd = end
        self.path = (self.x_pos, self.pathEnd)
        self.hurtTime = 0
        self.immune = False
        self.visible = True
        self.maxHealth = health
        self.currHealth = 1 + self.maxHealth - 1
        self.aitype = aitype
        self.aggressiveTime = 0
        self.aggressive = False

    def draw(self):
        if box.alive:
            self.movement()
        pygame.draw.rect(window, self.color, (self.x_pos, self.y_pos, self.width,
self.height))  # enemy
        pygame.draw.rect(window, (255, 0, 0), (self.x_pos, self.y_pos - 10,
self.width, 4))  # health bar red
        pygame.draw.rect(window, (0, 255, 0), (
            self.x_pos, self.y_pos - 10, (self.currHealth / self.maxHealth) *
```

```
self.width, 4))  # health bar green

    def movement(self):
        if self.aitype == 1 and not self.aggressive:
            if not self.immune:
                if self.vel > 0:
                    if self.x_pos + self.vel + self.width < self.path[1]:
                        self.x_pos += self.vel
                    else:
                        self.vel *= -1
                else:
                    if self.x_pos + self.vel > self.path[0]:
                        self.x_pos += self.vel
                    else:
                        self.vel *= -1

        elif self.aitype == 2 or self.aggressive:
            if not self.immune:
                if self.x_pos < box.x_pos:
                    self.x_pos += self.vel
                else:
                    self.x_pos -= self.vel
            if not box.alive:
                self.aitype = 1

    def bullet_hit(self):
        if self.currHealth - 1 > 0:
            sounds.enemyHurt.play()
        self.hurtTime = 1
        self.currHealth -= 1
        if self.aitype == 1:
            if random.randint(0, 100) % 5 == 0 or self.aggressive:
                self.aggressiveTime = 1


def redraw_game_window():
    global backgroundColor
    background = pygame.image.load('unnamed.png')
    text = font.render('Score:' + str(box.score), False, (255, 255, 255))
    hs = font.render('High Score:' + str(high_score), False, (255, 255, 255))
    window.blit(background, (0, 0))
    for beatBox in enemies:
        if beatBox.visible:
            beatBox.draw()
    for bullet in bullets:
        bullet.draw()
    if box.alive:
        if box.visible:
            box.draw()
        window.blit(text, (10, 10))
```

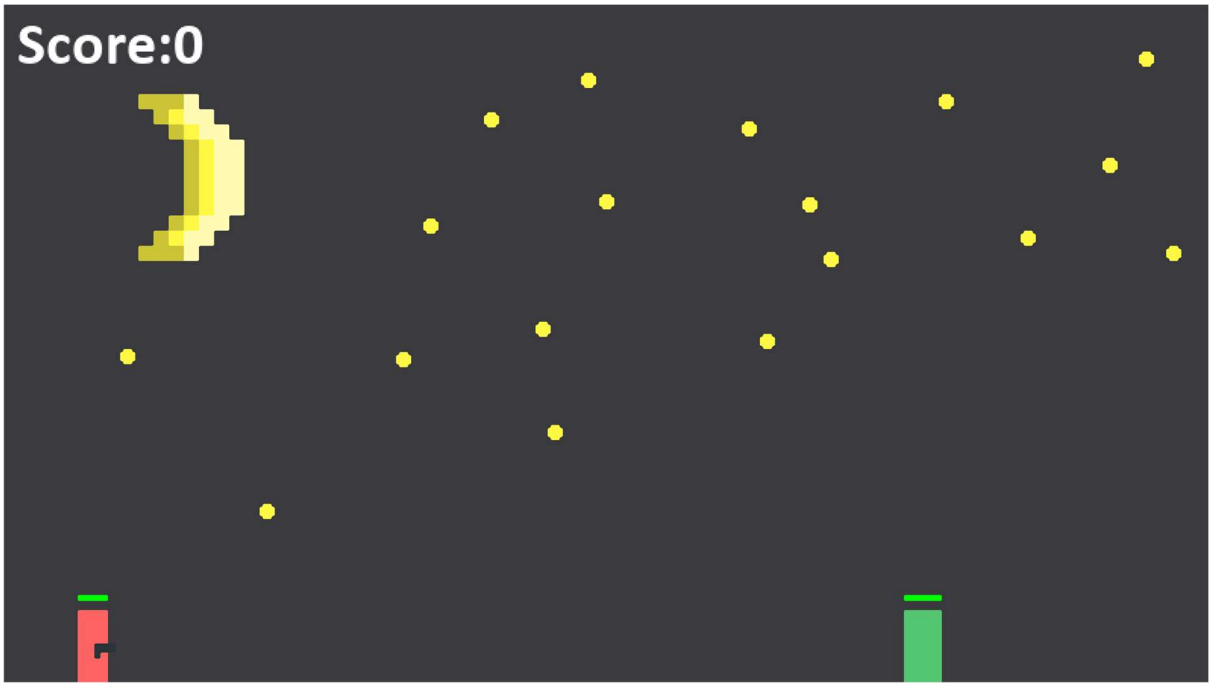
```
    if not box.alive:
        gameOver = pygame.font.Font('font.ttf', 70).render('Game Over', False,
(255, 255, 255))
        window.blit(gameOver,
                    (window_width // 2 - gameOver.get_width() // 2, window_height
// 2 - gameOver.get_height()))
        window.blit(text, (window_width // 2 - text.get_width() // 2,
window_height // 2 + text.get_height()))
        window.blit(hs, (window_width // 2 - hs.get_width() // 2, window_height //
2 + 3*text.get_height()))
    main_window.blit(pygame.transform.scale(window,(1600, 900)),(0, 0))
    pygame.display.update()


def basic_controls():
    global lastEnemy
    box.movement()

    if box.currHealth <= 0:
        box.alive = False

    for bullet in bullets:
        if bullet.x_pos < window_width and bullet.x_pos > 0:
            bullet.x_pos += bullet.vel
        else:  # bullet is off screen kill it
            bullets.pop(bullets.index(bullet))

    if (keys[pygame.K_DOWN] or keys[pygame.K_SPACE] or keys[pygame.K_s]) and not
box.reloading and box.alive:
        sounds.gunShoot.play()
        if box.direction > 0:
            bullets.append(
                projectile(round(box.x_pos + box.width), round(box.y_pos +
(box.height // 2)), box.direction))
        else:
            bullets.append(projectile(round(box.x_pos), round(box.y_pos +
(box.height // 2)), box.direction))
        box.curr_ammo -= 1

    if not box.curr_ammo > 0:
        box.lastReloadTime = time.perf_counter()
        box.reloading = True
        box.curr_ammo = box.max_ammo

    if box.reloading and time.perf_counter() - box.lastReloadTime > 2:
        box.reloading = False
        sounds.reloaded = False

    if box.reloading and time.perf_counter() - box.lastReloadTime > 0.5:
        if not sounds.reloaded:
```

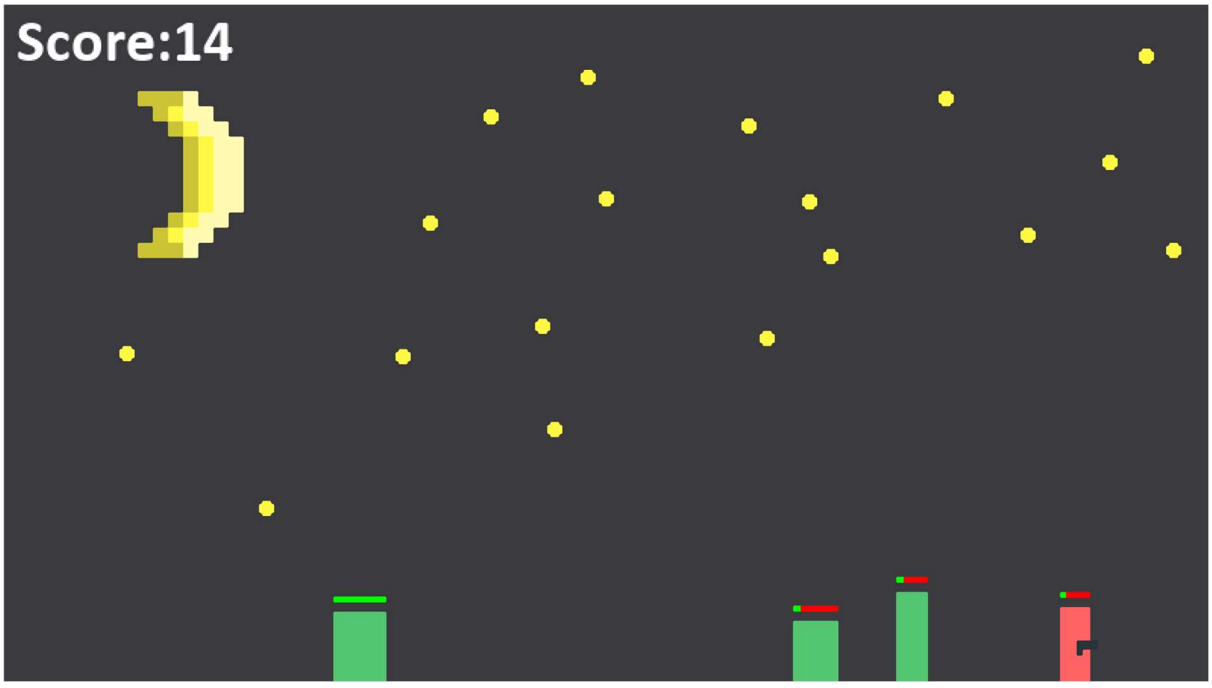
```
            sounds.gunReload.play()
            sounds.reloaded = True

    if box.immune:
        if box.hurtTime % 2 == 0:
            box.color = 255, 0, 0
            box.visible = True
        else:
            box.visible = False
    else:
        box.color = 255, 100, 100
        box.visible = True

    for beatBox in enemies:
        if beatBox.immune:
            if beatBox.hurtTime % 2 == 0:
                beatBox.color = 255, 0, 0
                beatBox.visible = True
            else:
                beatBox.visible = False
        else:
            beatBox.color = 84, 197, 113
            beatBox.visible = True

        if beatBox.currHealth <= 0:
            sounds.enemyDeath.play()
            box.score += 1
            enemies.pop(enemies.index(beatBox))

    if box.alive:
        if len(enemies) < box.score // 5 + 1:
            spawnEnemy()
            lastEnemy = time.perf_counter()
        elif len(enemies) <= box.score * 10:
            if time.perf_counter() - lastEnemy > random.randint(30, 60):
                spawnEnemy()
                lastEnemy = time.perf_counter()


def check_contact():
    for beatBox in enemies:
        if not beatBox.immune:
            for bullet in bullets:
                if bullet.x_pos < beatBox.x_pos + beatBox.width and bullet.x_pos +
bullet.radius > beatBox.x_pos:
                    if bullet.y_pos < beatBox.y_pos + beatBox.height and
bullet.y_pos + bullet.radius > beatBox.y_pos:
                        beatBox.bullet_hit()
                        bullets.pop(bullets.index(bullet))
```

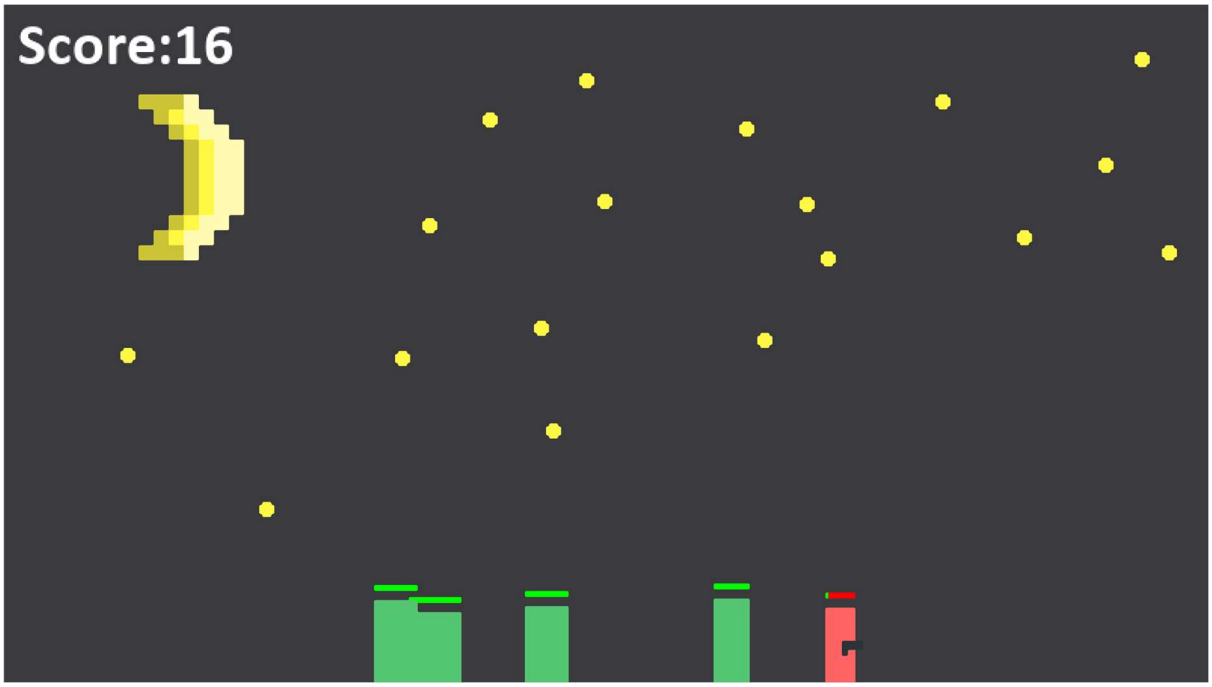
```
        if not box.immune and not beatBox.immune and box.alive:
            if box.x_pos < beatBox.x_pos + beatBox.width and box.x_pos + box.width
> beatBox.x_pos:
                if box.y_pos < beatBox.y_pos + beatBox.height and box.y_pos +
box.height > beatBox.y_pos:
                    box.enemy_touch()


def clocks():
    if box.hurtTime > 0:
        if box.hurtTime > 10:
            box.hurtTime = 0
            box.immune = False
        else:
            box.immune = True
            box.hurtTime += 1

    for beatBox in enemies:
        if beatBox.hurtTime > 0:
            if beatBox.hurtTime > 10:
                beatBox.hurtTime = 0
                beatBox.immune = False
            else:
                beatBox.immune = True
                beatBox.hurtTime += 1

    for beatBox in enemies:
        if beatBox.hurtTime == 0:
            if beatBox.aggressiveTime > 0:
                if beatBox.aggressiveTime > 50:
                    beatBox.aggressiveTime = 0
                    beatBox.aggressive = False
                else:
                    beatBox.aggressive = True
                    beatBox.aggressiveTime += 1


def spawnEnemy():
    goodCoordinates = False
    h = random.randint(40, 60)
    w = random.randint(15, 35)
    hel = round((w / 35) * 7)
    temp_var = random.randint(0, 10) % 3
    if temp_var == 2:
        aitype = 2
    else:
        aitype = 1
    while not goodCoordinates:
        if aitype == 1:
            x = random.randint(0, window_width - 200)
```

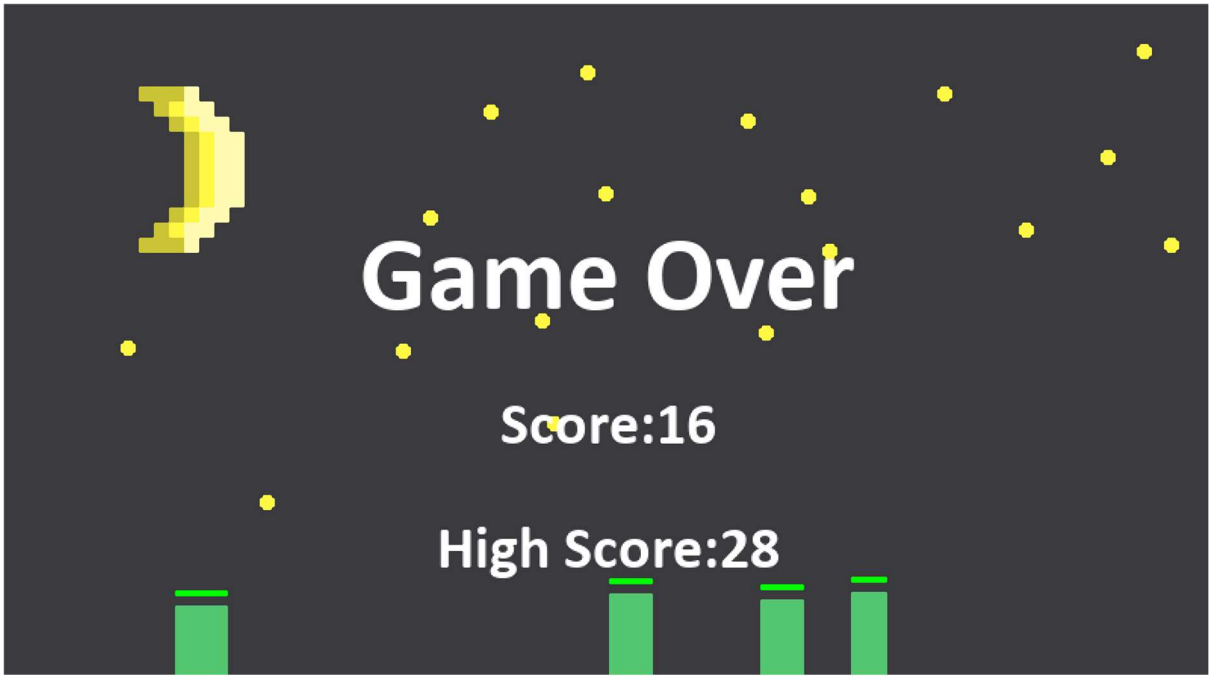
```
            end = random.randint(x, window_width - 50)
            if end - x > 200 and abs(box.x_pos - x) > 100:
                goodCoordinates = True
        else:
            x = random.randint(0, window_width - w)
            end = window_width - 50

            if abs(box.x_pos - x) > 100:
                goodCoordinates = True
    global enemies
    enemies.append(enemy(x, 450 - h, end, w, h, hel, aitype))

box = player(50, 400, 20, 50, 10)
bullets = []
font = pygame.font.Font("font.ttf", 40)
enemies = [enemy(350, 400, 700, 25, 50, 2, 1)]
lastEnemy = time.perf_counter()
pygame.mixer.music.play(-1)
clock = pygame.time.Clock()
high_score  = int(open('save.dat','r').read())

# main program
run = True
while run:
    pygame.time.delay(100)
    keys = pygame.key.get_pressed()

    for event in pygame.event.get():

        # Quit
        if event.type == pygame.QUIT:
            print("See You Later")
            run = False

    basic_controls()
    check_contact()
    clocks()
    redraw_game_window()

    if box.score > high_score:
        high_score = box.score

    clock.tick(120)


if int(box.score) >= int(high_score):
    f = open('save.dat','w')
    f.write(str(box.score))
# end
pygame.quit()
```

Output






Score:16


Game Over
Score:16
High Score:28

Error and its Types

An error, sometime called “A BUG” is anything in the code that prevents a program from compiling and running correctly. There are broadly three types of errors as follows:

1. Compile- time errors: Errors that occurs during compilation of a program is called compile time error. It has two types as follows:

    a. Syntax error: It refers to formal rules governing the construction of valid statements in a language.

    b. Semantics error: It refers to the set of rules which give the meaning of a statement.

2. Run time Errors: Errors that occur during the execution of program are run time errors. These are harder to detect errors. Some run-time error stop the execution of program which is then called program “Crashed”.

3. Logical Errors: Sometimes, even if you don’t encounter any error during compiling-time and runtime, your program does not provide the correct result. This is because of the programmer’s mistaken analysis of the problem he or she is trying to solve. Such errors are called logical error.

Testing

Alpha Testing: It is the most common type of testing used in the software industry. The objective of this testing is to identify all possible issues or defects before releasing it into the market or to the user. It is conducted at the developer's site.

Beta Testing: It is a formal type of software testing which is carried out by the customers. It is performed in a real environment before releasing the products into the market for the actual end-users. It is carried out to ensure that there are no major failures in the software or product and it satisfies the business requirement. Beta Testing is successful when the customer accepts the software.

White Box Testing: White box testing is based on the knowledge about the internal logic of an application's code. It is also known as Glass box Testing. Internal Software and code working should be known for performing this type of testing. These tests are based on the coverage of the code statements, branches, paths, conditions etc.

Black Box Testing: It is a software testing, method in which the internal structure or design of the item to be tested is not known to the tester. This method of testing can be applied virtually to every level of the software testing.

Maintenance

Programming maintenance refers to the modifications in the program. After it has been completed, in order to meet changing requirement or to take care of the errors that shown up. There are four types of maintenance:

Corrective Maintenance: When the program after compilation shows error because of some unexpected situations, untested areas such errors are fixed up by Corrective maintenance.

Adaptive Maintenance: Changes in the environment in which an information system operates may lead to system management. To accommodate changing needs time to time maintenance is done and is called Adaptive maintenance.

Preventive Maintenance: If possible the errors could be anticipated before they actually occur; the maintenance is called Preventive maintenance.

Perfective Maintenance: In this rapidly changing world, information technology is the fastest growing area. If te existing system is maintained to keep tuned with the new features, new facilities, new capabilities, it is said to be Perfective maintenance.

Appendix

Module: pygame – package for 2d game development

Functions	Working
`init()`	Staets up pygame
`surface()`	Creates a blank canvas to draw objects on
`blit()`	Shows pixels on screen
`mixer()`	Used for adding sounds and music
`event.get()`	To check if a key or button is pressed
`transform.scale2x()`	Scales a surface to twice its original size
`draw.circle()`	Draws a circle on the screen
`draw.rect()`	Draws a rectangle on the screen
`image.load()`	Loads an image which can be shows on screen later with blit()
`font.render()`	To save text as image which can be later shown

Bibliography

Coding Help:

Stackoverflow

Youtube: TheNewBoston

www.pygame.org

Informatics Practices by Sumita Arora

Computer Science Teacher

Music:

Chrome Music Lab