Enemies Destroyer Game

Introduction

In this game stop the enemies invasion with a soldier. Fun for all with additional features to help children with special needs, this game can be played with a tap anywhere on the screen or by using a single external switch to zap the enemies. Clear the screen ready for the next wave.

The enemies destroyer game is a very attractive game for any ages peoples. The robot moves around the world and protect world from various enemies. Enemies attacks the robot. The robot shoots and destroy his enemies and save world. The point are being increased automatically by destroying each enemies. As the first-person shooter has evolved to be bloated in terms of costs and production requirements, its game play mechanics have atrophied over the years.

Objectives of the Gaming System

The game is developed for full-time entertainment . It teaches the Gamer to be alert at every situation he/she faces, because if the Gamer is not fully alert and notice the saucer fire he/she must be hit by the saucer-bombs. Though the proposed game is an action game, it doesn't involve direct violence. So it can also be viewed as a non violence game. Kids can also play this game, because the design of the game is very simple, controlling the game is very easy .The purpose of this research is to provide a virtual image for the combination of both structured and unstructured information of my project "Enemies Destroyer ". This is a single-player strategy game on the Windows platform. The player will progress through levels which require precise manipulation of the environment, though the game encourages creativity. The episodic structure of the game facilitates the pace of the story. I demonstrate the action flow between inputs, display.

Activity Diagram:

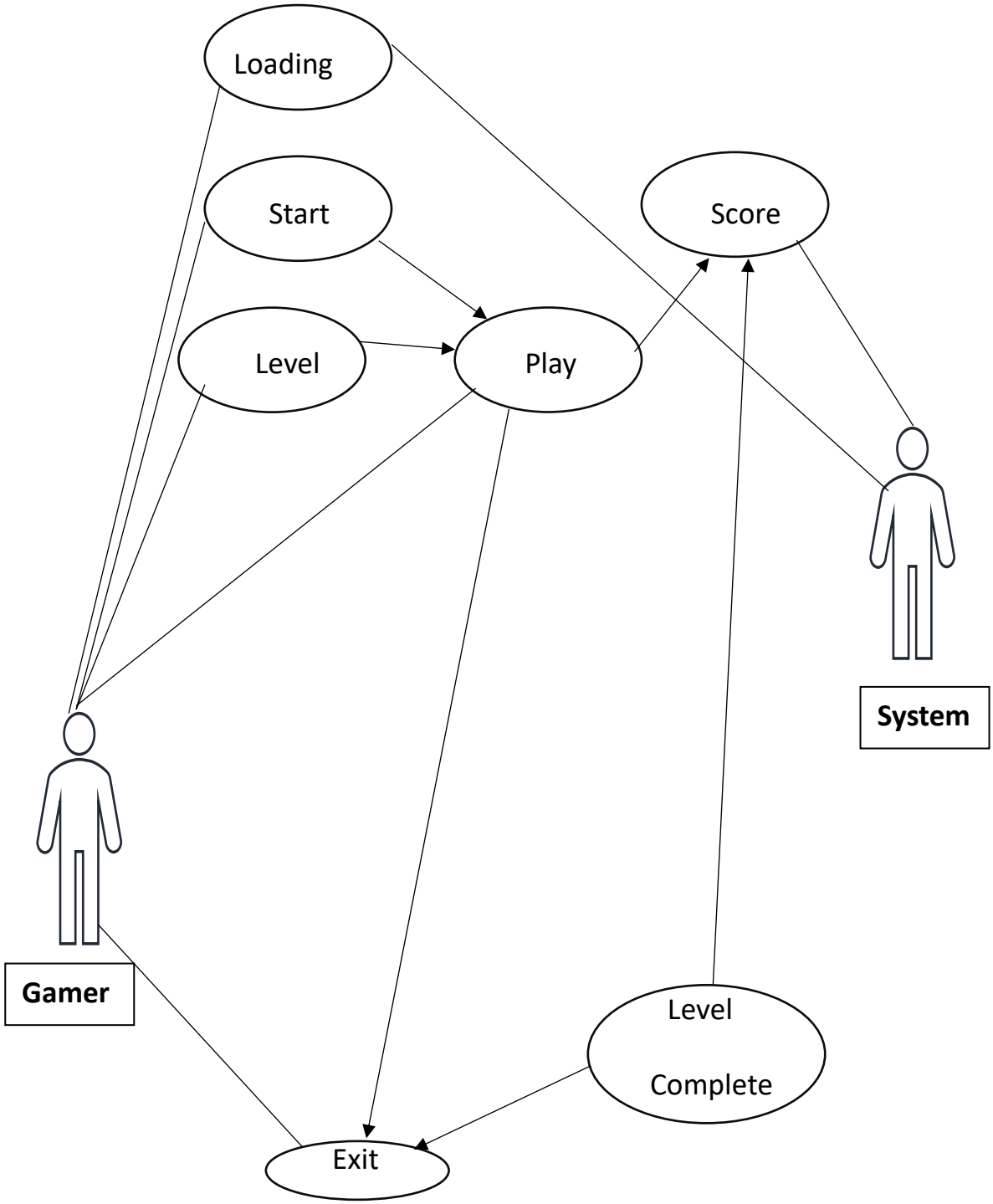


Fig1.1:Activity diagram

In Fig 1.1 here when the gamer start to play the game .The gamer will see the interface of loading page .In the loading page there are two opposition one is start button and another is exit button. If the gamer want to play game then he or she press the start function and if he do not want to play then press the exit button to leave the game. The game start and the soldier have to kill the enemies as following the gaming rules .The gaming system will count the enemies killing number and the life box number and bullet ,grandees number. After that completing the level the gamer will go to the next level and there are three level. If the complete all the level or die the score will show in the display and in that interface there are also a restart button if the player want to play again the system will take into the game playing interface and start from the first.

Gaming Details

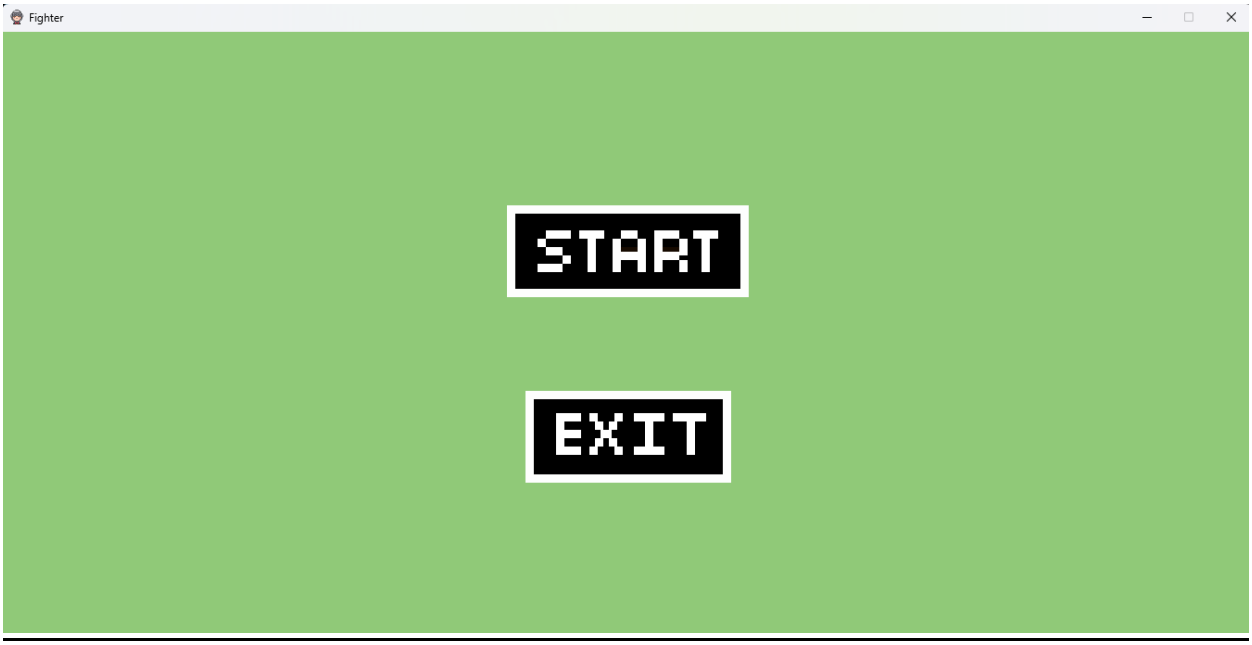


Fig1.2:Loading Page

1.Game Loading Interface: When user open the game, there will be a Loading interface. Where will be two button . One is start button and another is exit button.

2.Start Game button: The Start Game button work as the way to the next interface for selecting different game level.

3.Exit button: There will be an Exit button to close the current playing game and also leave the game.

Fig1.3:Game Start Interface

4.Game Start Interface: Here the soldier will kill the enemies using bullets or grenades. Also there is a life box .

5.Soldier control: Soldier will be control by the "Left end Right" using A and D in the keyboard.

Fig1.4:Bullet,Grenades &Life Box

6.Bullet, Grenades &Life Box: Here we add bullets grenades and life box .If we want to defeat or kill the enemies the soldiers will use the bullet and grenades. On the other hand, there is also a life box. The full green color is life and the red color is life decreased .If the soldier will injured by the enemies the life time will be decreased. Bullets and grenades are also being count.

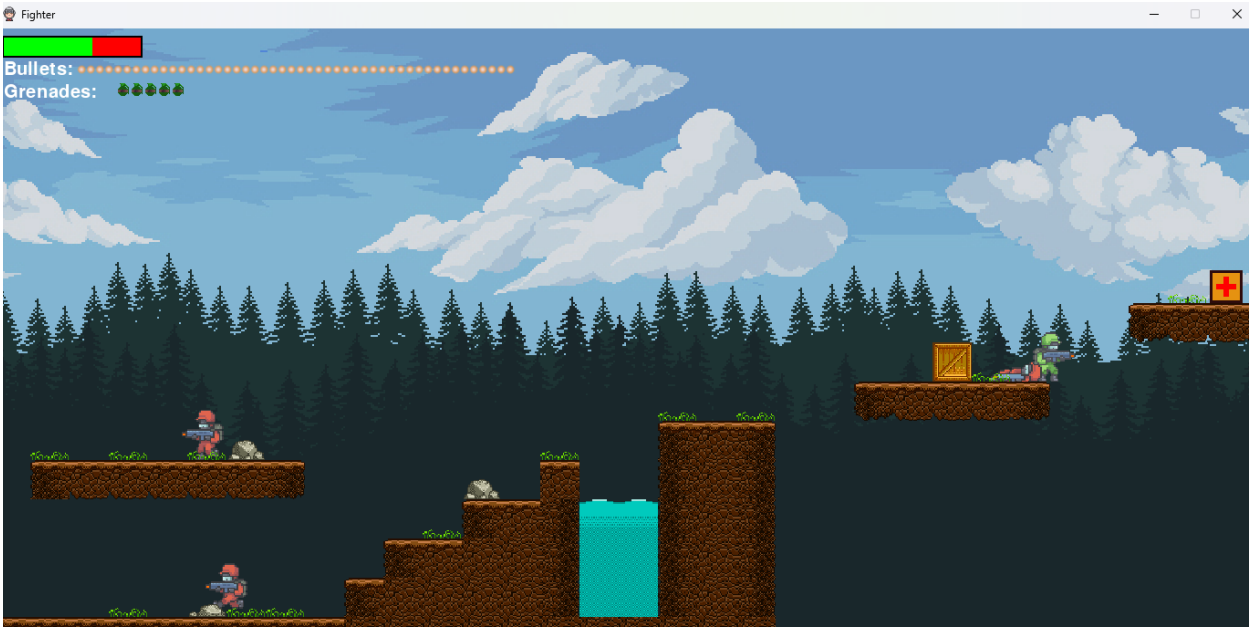


Fig1.5: Taking Life Box Add, Bullet Box, Grenade Box

7.Taking Life, Bullet, Grenade Box Add: Using all this box we can increase our life and bullets and grenades. This all are increase one.

8.Level selection: There will be three different level in the game. Those Level will define on the basis of the speed.

Fig1.6:Restart & Score

9.Restart & Score: After dying the soldiers there will come a restart and score interface. Where how many enemies are killed by the soldier the score will be show and if anyone want to play that more they can play it pressing the restart button and it will be back to the start game interface.

Enemy Destroyer Code:

```
import pygame
from pygame import mixer
import os
import random
import csv
import button

mixer.init()
pygame.init()


SCREEN_WIDTH = 1385
SCREEN_HEIGHT = float(SCREEN_WIDTH * 0.50)

screen = pygame.display.set_mode((SCREEN_WIDTH, SCREEN_HEIGHT))
screen = pygame.display.set_mode((SCREEN_WIDTH, SCREEN_HEIGHT))
pygame.display.set_caption('Enemies Destroyer')
icon = pygame.image.load('icon.png')
pygame.display.set_icon(icon)
start_game = False

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

GRAVITY = 0.75
SCROLL_THRESH = 200
ROWS = 16
COLS = 150
TILE_SIZE = SCREEN_HEIGHT // ROWS
TILE_TYPES = 21
MAX_LEVELS = 3
screen_scroll = 0
bg_scroll = 0
level = 1
start_game = False
start_intro = False
```

```
moving_left = False
moving_right = False
shoot = False
grenade = False
grenade_thrown = False

jump_fx = pygame.mixer.Sound('audio/jump.wav')
jump_fx.set_volume(0.05)
shot_fx = pygame.mixer.Sound('audio/shot.wav')
shot_fx.set_volume(0.05)
grenade_fx = pygame.mixer.Sound('audio/grenade.wav')
grenade_fx.set_volume(0.05)



start_img = pygame.image.load('img/start_btn.png').convert_alpha()
exit_img = pygame.image.load('img/exit_btn.png').convert_alpha()
restart_img = pygame.image.load('img/restart_btn.png').convert_alpha()

pine1_img = pygame.image.load('img/Background/pine1.png').convert_alpha()
pine2_img = pygame.image.load('img/Background/pine2.png').convert_alpha()
mountain_img = pygame.image.load('img/Background/mountain.png').convert_alpha()
sky_img = pygame.image.load('img/Background/sky_cloud.png').convert_alpha()

img_list = []
for x in range(TILE_TYPES):
    img = pygame.image.load(f'img/Tile/{x}.png')
    img = pygame.transform.scale(img, (TILE_SIZE, TILE_SIZE))
    img_list.append(img)

bullet_img = pygame.image.load('img/icons/bullet.png').convert_alpha()

grenade_img = pygame.image.load('img/icons/grenade.png').convert_alpha()

health_box_img = pygame.image.load('img/icons/health_box.png').convert_alpha()
ammo_box_img = pygame.image.load('img/icons/ammo_box.png').convert_alpha()
grenade_box_img = pygame.image.load('img/icons/grenade_box.png').convert_alpha()
item_boxes = {
    'Health'    : health_box_img,
    'Ammo'      : ammo_box_img,
    'Grenade'   : grenade_box_img
}
```

```
BG = (144, 201, 120)
RED = (255, 0, 0)
WHITE = (255, 255, 255)
GREEN = (0, 255, 0)
BLACK = (0, 0, 0)
PINK = (235, 65, 54)

font = pygame.font.SysFont('Futura', 30)

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


def draw_bg():
    screen.fill(BG)
    width = sky_img.get_width()
    for x in range(5):
        screen.blit(sky_img, ((x * width) - bg_scroll * 0.5, 0))
                screen.blit(pine1_img, ((x * width) - bg_scroll * 0.7, SCREEN_HEIGHT -
pine1_img.get_height() - 150))
                screen.blit(pine2_img, ((x * width) - bg_scroll * 0.8, SCREEN_HEIGHT -
pine2_img.get_height()))



def reset_level():
    enemy_group.empty()
    bullet_group.empty()
    grenade_group.empty()
    explosion_group.empty()
    item_box_group.empty()
    decoration_group.empty()
    water_group.empty()
    exit_group.empty()


    data = []
    for row in range(ROWS):
        r = [-1] * COLS
        data.append(r)
```

```
    return data



class Soldier(pygame.sprite.Sprite):
    def __init__(self, char_type, x, y, scale, speed, ammo, grenades):
        pygame.sprite.Sprite.__init__(self)
        self.alive = True
        self.char_type = char_type
        self.speed = speed
        self.ammo = ammo
        self.start_ammo = ammo
        self.shoot_cooldown = 0
        self.grenades = grenades
        self.health = 100
        self.max_health = self.health
        self.direction = 1
        self.vel_y = 0
        self.jump = False
        self.in_air = True
        self.flip = False
        self.animation_list = []
        self.frame_index = 0
        self.action = 0
        self.update_time = pygame.time.get_ticks()

        self.move_counter = 0
        self.vision = pygame.Rect(0, 0, 150, 20)
        self.idling = False
        self.idling_counter = 0


        animation_types = ['Idle', 'Run', 'Jump', 'Death']
        for animation in animation_types:

            temp_list = []

            num_of_frames = len(os.listdir(f'img/{self.char_type}/{animation}'))
            for i in range(num_of_frames):
                img = pygame.image.load(f'img/{self.char_type}/{animation}/{i}.png').convert_alpha()
                img = pygame.transform.scale(img, (int(img.get_width() * scale), int(img.get_height() *
scale)))
```

```
            temp_list.append(img)
        self.animation_list.append(temp_list)

    self.image = self.animation_list[self.action][self.frame_index]
    self.rect = self.image.get_rect()
    self.rect.center = (x, y)
    self.width = self.image.get_width()
    self.height = self.image.get_height()


def update(self):
    self.update_animation()
    self.check_alive()

    if self.shoot_cooldown > 0:
        self.shoot_cooldown -= 1


def move(self, moving_left, moving_right):

    screen_scroll = 0
    dx = 0
    dy = 0


    if moving_left:
        dx = -self.speed
        self.flip = True
        self.direction = -1
    if moving_right:
        dx = self.speed
        self.flip = False
        self.direction = 1

    if self.jump == True and self.in_air == False:
        self.vel_y = -11
        self.jump = False
        self.in_air = True

    self.vel_y += GRAVITY
    if self.vel_y > 10:
        self.vel_y
    dy += self.vel_y
```

```
        for tile in world.obstacle_list:

            if tile[1].colliderect(self.rect.x + dx, self.rect.y, self.width, self.height):
                dx = 0

                if self.char_type == 'enemy':
                    self.direction *= -1
                    self.move_counter = 0

            if tile[1].colliderect(self.rect.x, self.rect.y + dy, self.width, self.height):

                if self.vel_y < 0:
                    self.vel_y = 0
                    dy = tile[1].bottom - self.rect.top

                elif self.vel_y >= 0:
                    self.vel_y = 0
                    self.in_air = False
                    dy = tile[1].top - self.rect.bottom


        if pygame.sprite.spritecollide(self, water_group, False):
            self.health = 0

        level_complete = False
        if pygame.sprite.spritecollide(self, exit_group, False):
            level_complete = True

        if self.rect.bottom > SCREEN_HEIGHT:
            self.health = 0

        if self.char_type == 'player':
            if self.rect.left + dx < 0 or self.rect.right + dx > SCREEN_WIDTH:
                dx = 0

        self.rect.x += dx
        self.rect.y += dy

        if self.char_type == 'player':
            if (self.rect.right > SCREEN_WIDTH - SCROLL_THRESH and bg_scroll < (world.level_length
    * TILE_SIZE) - SCREEN_WIDTH)\
                    or (self.rect.left < SCROLL_THRESH and bg_scroll > abs(dx)):
                self.rect.x -= dx
```

```
            screen_scroll = -dx

        return screen_scroll, level_complete



    def shoot(self):
        if self.shoot_cooldown == 0 and self.ammo > 0:
            self.shoot_cooldown = 20
            bullet = Bullet(self.rect.centerx + (0.75 * self.rect.size[0] * self.direction), self.rect.centery,
self.direction)
            bullet_group.add(bullet)
            self.ammo -= 1
            shot_fx.play()


    def ai(self):
        if self.alive and player.alive:
            if self.idling == False and random.randint(1, 200) == 1:
                self.update_action(0)
                self.idling = True
                self.idling_counter = 50
            if self.vision.colliderect(player.rect):
                self.update_action(0)
                self.shoot()
            else:
                if self.idling == False:
                    if self.direction == 1:
                        ai_moving_right = True
                    else:
                        ai_moving_right = False
                    ai_moving_left = not ai_moving_right
                    self.move(ai_moving_left, ai_moving_right)
                    self.update_action(1)
                    self.move_counter += 1
                    self.vision.center = (self.rect.centerx + 75 * self.direction, self.rect.centery)

                    if self.move_counter > TILE_SIZE:
                        self.direction *= -1
                        self.move_counter *= -1
                else:
                    self.idling_counter -= 1
                    if self.idling_counter <= 0:
                        self.idling = False
```

```
        self.rect.x += screen_scroll

    def update_animation(self):

        ANIMATION_COOLDOWN = 100

        self.image = self.animation_list[self.action][self.frame_index]

        if pygame.time.get_ticks() - self.update_time > ANIMATION_COOLDOWN:
            self.update_time = pygame.time.get_ticks()
            self.frame_index += 1

        if self.frame_index >= len(self.animation_list[self.action]):
            if self.action == 3:
                self.frame_index = len(self.animation_list[self.action]) - 1
            else:
                self.frame_index = 0



    def update_action(self, new_action):

        if new_action != self.action:
            self.action = new_action

            self.frame_index = 0
            self.update_time = pygame.time.get_ticks()



    def check_alive(self):
        if self.health <= 0:
            self.health = 0
            self.speed = 0
            self.alive = False
            self.update_action(3)


    def draw(self):
        screen.blit(pygame.transform.flip(self.image, self.flip, False), self.rect)


class World():
```

```
def __init__(self):
    self.obstacle_list = []

def process_data(self, data):
    self.level_length = len(data[0])

    for y, row in enumerate(data):
        for x, tile in enumerate(row):
            if tile >= 0:
                img = img_list[tile]
                img_rect = img.get_rect()
                img_rect.x = x * TILE_SIZE
                img_rect.y = y * TILE_SIZE
                tile_data = (img, img_rect)
                if tile >= 0 and tile <= 8:
                    self.obstacle_list.append(tile_data)
                elif tile >= 9 and tile <= 10:
                    water = Water(img, x * TILE_SIZE, y * TILE_SIZE)
                    water_group.add(water)
                elif tile >= 11 and tile <= 14:
                    decoration = Decoration(img, x * TILE_SIZE, y * TILE_SIZE)
                    decoration_group.add(decoration)
                elif tile == 15:
                    player = Soldier('player', x * TILE_SIZE, y * TILE_SIZE, 1.65, 5, 20, 5)
                    health_bar = HealthBar(10, 10, player.health, player.health)
                elif tile == 16:
                    enemy = Soldier('enemy', x * TILE_SIZE, y * TILE_SIZE, 1.65, 2, 20, 0)
                    enemy_group.add(enemy)
                elif tile == 17:
                    item_box = ItemBox('Ammo', x * TILE_SIZE, y * TILE_SIZE)
                    item_box_group.add(item_box)
                elif tile == 18:
                    item_box = ItemBox('Grenade', x * TILE_SIZE, y * TILE_SIZE)
                    item_box_group.add(item_box)
                elif tile == 19:
                    item_box = ItemBox('Health', x * TILE_SIZE, y * TILE_SIZE)
                    item_box_group.add(item_box)
                elif tile == 20:
                    exit = Exit(img, x * TILE_SIZE, y * TILE_SIZE)
                    exit_group.add(exit)

    return player, health_bar
```

```
    def draw(self):
        for tile in self.obstacle_list:
            tile[1][0] += screen_scroll
            screen.blit(tile[0], tile[1])


class Decoration(pygame.sprite.Sprite):
    def __init__(self, img, x, y):
        pygame.sprite.Sprite.__init__(self)
        self.image = img
        self.rect = self.image.get_rect()
        self.rect.midtop = (x + TILE_SIZE // 2, y + (TILE_SIZE - self.image.get_height()))

    def update(self):
        self.rect.x += screen_scroll


class Water(pygame.sprite.Sprite):
    def __init__(self, img, x, y):
        pygame.sprite.Sprite.__init__(self)
        self.image = img
        self.rect = self.image.get_rect()
        self.rect.midtop = (x + TILE_SIZE // 2, y + (TILE_SIZE - self.image.get_height()))

    def update(self):
        self.rect.x += screen_scroll

class Exit(pygame.sprite.Sprite):
    def __init__(self, img, x, y):
        pygame.sprite.Sprite.__init__(self)
        self.image = img
        self.rect = self.image.get_rect()
        self.rect.midtop = (x + TILE_SIZE // 2, y + (TILE_SIZE - self.image.get_height()))

    def update(self):
        self.rect.x += screen_scroll


class ItemBox(pygame.sprite.Sprite):
    def __init__(self, item_type, x, y):
        pygame.sprite.Sprite.__init__(self)
        self.item_type = item_type
        self.image = item_boxes[self.item_type]
        self.rect = self.image.get_rect()
```

```
        self.rect.midtop = (x + TILE_SIZE // 2, y + (TILE_SIZE - self.image.get_height()))


    def update(self):
        #scroll
        self.rect.x += screen_scroll
        #check if the player has picked up the box
        if pygame.sprite.collide_rect(self, player):
            #check what kind of box it was
            if self.item_type == 'Health':
                player.health += 70
                if player.health > player.max_health:
                    player.health = player.max_health
            elif self.item_type == 'Ammo':
                player.ammo += 30
            elif self.item_type == 'Grenade':
                player.grenades += 8
            #delete the item box
            self.kill()


class HealthBar():
    def __init__(self, x, y, health, max_health):
        self.x = x
        self.y = y
        self.health = health
        self.max_health = max_health

    def draw(self, health):

        self.health = health
        ratio = self.health / self.max_health
        pygame.draw.rect(screen, BLACK, (self.x - 2, self.y - 2, 154, 24))
        pygame.draw.rect(screen, RED, (self.x, self.y, 150, 20))
        pygame.draw.rect(screen, GREEN, (self.x, self.y, 150 * ratio, 20))


class Bullet(pygame.sprite.Sprite):
    def __init__(self, x, y, direction):
        pygame.sprite.Sprite.__init__(self)
        self.speed = 20
        self.image = bullet_img
        self.rect = self.image.get_rect()
        self.rect.center = (x, y)
```

```
        self.direction = direction

    def update(self):

        self.rect.x += (self.direction * self.speed) + screen_scroll

        if self.rect.right < 0 or self.rect.left > SCREEN_WIDTH:
            self.kill()

        for tile in world.obstacle_list:
            if tile[1].colliderect(self.rect):
                self.kill()

        if pygame.sprite.spritecollide(player, bullet_group, False):
            if player.alive:
                player.health -= 5
                self.kill()
        for enemy in enemy_group:
            if pygame.sprite.spritecollide(enemy, bullet_group, False):
                if enemy.alive:
                    enemy.health -= 50
                    self.kill()



class Grenade(pygame.sprite.Sprite):
    def __init__(self, x, y, direction):
        pygame.sprite.Sprite.__init__(self)
        self.timer = 100
        self.vel_y = -11
        self.speed = 7
        self.image = grenade_img
        self.rect = self.image.get_rect()
        self.rect.center = (x, y)
        self.width = self.image.get_width()
        self.height = self.image.get_height()
        self.direction = direction

    def update(self):
        self.vel_y += GRAVITY
        dx = self.direction * self.speed
        dy = self.vel_y
```

```
        for tile in world.obstacle_list:

            if tile[1].colliderect(self.rect.x + dx, self.rect.y, self.width, self.height):
                self.direction *= -1
                dx = self.direction * self.speed

            if tile[1].colliderect(self.rect.x, self.rect.y + dy, self.width, self.height):
                self.speed = 0

                if self.vel_y < 0:
                    self.vel_y = 0
                    dy = tile[1].bottom - self.rect.top

                elif self.vel_y >= 0:
                    self.vel_y = 0
                    dy = tile[1].top - self.rect.bottom



        self.rect.x += dx + screen_scroll
        self.rect.y += dy


        self.timer -= 3
        if self.timer <= 3:
            self.kill()
            grenade_fx.play()
            explosion = Explosion(self.rect.x, self.rect.y, 3)
            explosion_group.add(explosion)

            if abs(self.rect.centerx - player.rect.centerx) < TILE_SIZE * 2 and \
                abs(self.rect.centery - player.rect.centery) < TILE_SIZE * 2:
                player.health -= 10
            for enemy in enemy_group:
                if abs(self.rect.centerx - enemy.rect.centerx) < TILE_SIZE * 2 and \
                    abs(self.rect.centery - enemy.rect.centery) < TILE_SIZE * 2:
                    enemy.health -= 50



class Explosion(pygame.sprite.Sprite):
    def __init__(self, x, y, scale):
        pygame.sprite.Sprite.__init__(self)
        self.images = []
```

```
        for num in range(1, 6):
            img = pygame.image.load(f'img/explosion/exp{num}.png').convert_alpha()
             img = pygame.transform.scale(img, (int(img.get_width() * scale), int(img.get_height() *
scale)))
            self.images.append(img)
        self.frame_index = 0
        self.image = self.images[self.frame_index]
        self.rect = self.image.get_rect()
        self.rect.center = (x, y)
        self.counter = 0


    def update(self):
        #scroll
        self.rect.x += screen_scroll

        EXPLOSION_SPEED = 5

        self.counter += 1

        if self.counter >= EXPLOSION_SPEED:
            self.counter = 0
            self.frame_index += 1
            if self.frame_index >= len(self.images):
                self.kill()
            else:
                self.image = self.images[self.frame_index]


class ScreenFade():
    def __init__(self, direction, colour, speed):
        self.direction = direction
        self.colour = colour
        self.speed = speed
        self.fade_counter = 0


    def fade(self):
        fade_complete = False
        self.fade_counter += self.speed
        if self.direction == 1:
            pygame.draw.rect(screen, self.colour, (0 - self.fade_counter, 0, SCREEN_WIDTH // 2,
SCREEN_HEIGHT))
```

```
            pygame.draw.rect(screen, self.colour, (SCREEN_WIDTH // 2 + self.fade_counter, 0, SCREEN_WIDTH, SCREEN_HEIGHT))
            pygame.draw.rect(screen, self.colour, (0, 0 - self.fade_counter, SCREEN_WIDTH, SCREEN_HEIGHT // 2))
            pygame.draw.rect(screen, self.colour, (0, SCREEN_HEIGHT // 2 +self.fade_counter, SCREEN_WIDTH, SCREEN_HEIGHT))
        if self.direction == 2:
            pygame.draw.rect(screen, self.colour, (0, 0, SCREEN_WIDTH, 0 + self.fade_counter))
        if self.fade_counter >= SCREEN_WIDTH:
            fade_complete = True

        return fade_complete


intro_fade = ScreenFade(1, BLACK, 4)
death_fade = ScreenFade(2, PINK, 4)


start_button = button.Button(SCREEN_WIDTH // 2 - 130, SCREEN_HEIGHT // 2 - 150, start_img, 1)
exit_button = button.Button(SCREEN_WIDTH // 2 - 110, SCREEN_HEIGHT // 2 + 50, exit_img, 1)
restart_button = button.Button(SCREEN_WIDTH // 150 - 15, SCREEN_HEIGHT // 2 - 150, restart_img, 1)

enemy_group = pygame.sprite.Group()
bullet_group = pygame.sprite.Group()
grenade_group = pygame.sprite.Group()
explosion_group = pygame.sprite.Group()
item_box_group = pygame.sprite.Group()
decoration_group = pygame.sprite.Group()
water_group = pygame.sprite.Group()
exit_group = pygame.sprite.Group()


world_data = []
for row in range(ROWS):
    r = [-1] * COLS
    world_data.append(r)

with open(f'level{level}_data.csv', newline='') as csvfile:
    reader = csv.reader(csvfile, delimiter=',')
    for x, row in enumerate(reader):
        for y, tile in enumerate(row):
            world_data[x][y] = int(tile)
```

```
world = World()
player, health_bar = world.process_data(world_data)



run = True
while run:

    clock.tick(FPS)

    if start_game == False:

        screen.fill(BG)

        if start_button.draw(screen):
            start_game = True
            start_intro = True
        if exit_button.draw(screen):
            run = False
    else:

        draw_bg()

        world.draw()

        health_bar.draw(player.health)

        draw_text('Bullets: ', font, WHITE, 10, 35)
        for x in range(player.ammo):
            screen.blit(bullet_img, (90 + (x * 10), 40))

        draw_text('Grenades: ', font, WHITE, 10, 60)
        for x in range(player.grenades):
            screen.blit(grenade_img, (135 + (x * 15), 60))


        player.update()
        player.draw()

        for enemy in enemy_group:
            enemy.ai()
            enemy.update()
            enemy.draw()
```

```
bullet_group.update()
grenade_group.update()
explosion_group.update()
item_box_group.update()
decoration_group.update()
water_group.update()
exit_group.update()
bullet_group.draw(screen)
grenade_group.draw(screen)
explosion_group.draw(screen)
item_box_group.draw(screen)
decoration_group.draw(screen)
water_group.draw(screen)
exit_group.draw(screen)


if start_intro == True:
    if intro_fade.fade():
        start_intro = False
        intro_fade.fade_counter = 0

if player.alive:

    if shoot:
        player.shoot()

    elif grenade and grenade_thrown == False and player.grenades > 0:
        grenade = Grenade(player.rect.centerx + (0.5 * player.rect.size[0] * player.direction),\
                    player.rect.top, player.direction)
        grenade_group.add(grenade)

        player.grenades -= 1
        grenade_thrown = True
    if player.in_air:
        player.update_action(2)
    elif moving_left or moving_right:
        player.update_action(1)
    else:
        player.update_action(0)
    screen_scroll, level_complete = player.move(moving_left, moving_right)
    bg_scroll -= screen_scroll

    if level_complete:
        start_intro = True
```

```
            level += 1
            bg_scroll = 0
            world_data = reset_level()
            if level <= MAX_LEVELS:
                with open(f'level{level}_data.csv', newline='') as csvfile:
                    reader = csv.reader(csvfile, delimiter=',')
                    for x, row in enumerate(reader):
                        for y, tile in enumerate(row):
                            world_data[x][y] = int(tile)
                world = World()
                player, health_bar = world.process_data(world_data)
        else:
            screen_scroll = 0
            if death_fade.fade():
                if restart_button.draw(screen):
                    death_fade.fade_counter = 0
                    start_intro = True
                    bg_scroll = 0
                    world_data = reset_level()
                    with open(f'level{level}_data.csv', newline='') as csvfile:
                        reader = csv.reader(csvfile, delimiter=',')
                        for x, row in enumerate(reader):
                            for y, tile in enumerate(row):
                                world_data[x][y] = int(tile)
                    world = World()
                    player, health_bar = world.process_data(world_data)


    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            run = False
        if event.type == pygame.KEYDOWN:
            if event.key == pygame.K_a:
                moving_left = True
            if event.key == pygame.K_d:
                moving_right = True
            if event.key == pygame.K_SPACE:
                shoot = True
            if event.key == pygame.K_q:
                grenade = True
            if event.key == pygame.K_w and player.alive:
                player.jump = True
                jump_fx.play()
            if event.key == pygame.K_ESCAPE:
```

```
            run = False
        if event.type == pygame.KEYUP:
            if event.key == pygame.K_a:
                moving_left = False
            if event.key == pygame.K_d:
                moving_right = False
            if event.key == pygame.K_SPACE:
                shoot = False
            if event.key == pygame.K_q:
                grenade = False
                grenade_thrown = False


    pygame.display.update()

pygame.quit()
```

Features

In this game there are some features that might be interesting for gamer who have participate this game.

First of all there are two button two button start and exit button and you have to select the start button to play this game.

After that you have to safe yourself from enemies the and kill the enemies to go next level. There are also life and bullet and grenades point.

If you die then there comes a restart interface and also show your score. If you want to play it again you have to press the restart button.

Target Population

Your game's target audience is the demographics and interests of the majority of it's players. This game tend to be the most popular console video games, and strategy and roleplaying games tend to be most popular non-console games. However, several survey and focus group studies suggest that the types of games that older gamers play, or would like to play, are different from the games that are most popular .This may not be surprising given the potential mismatch between the

visual, attentional, and processing speed demands of popular action, sports, and strategy games and older adults' poorer perceptual/cognitive abilities. Unfortunately, older adult game preference and gaming habits have been a relatively understudied topic. Additional research is needed to better understand older adults' motivation to engage in game play and predictors of game preference. Digital gaming represents a novel domain with which to explore and validate new and existing models of technology adoption and adherence.

Awareness & Social Economics Benefit

In this game it give a social message that if any enemies want to destroy our country we have to save our country as like the robot. We always have to be careful from the enemies. This game also give entertainment. This game is played by all ages people. The social and economic impacts of gaming—with each data source providing only part of the "impact picture". There remains significant challenges to achieving a holistic understanding of the effects of gaming. Principal among these challenges is the need to collect and analyse objective evidence. Poor information availability and measurement concerns impose other complicating factors. A model capable of organising and integrating the diverse findings, levels of analysis, samples and methods found in gaming impact research is required if informed decisions about the "benefits and costs of gaming" and "future directions for gaming research" are to be made. That is, a holistic understanding of gaming effects would be aided at this point by the development of a comprehensive framework that supports the integration and evaluation of current research findings and methods in the gaming literature. This project aims to create such a framework.

Conclusion

The enemies destroyer game is a very attractive game for any ages peoples. The gamer will enjoy this game very much. Computer games offer opportunities for learning, whether it be improving reaction time, learning more about ancient civilizations, or rehearsing social skills or problem. Games can both teach skills and allow rehearsal or integration of these skills

REFERENCES

[1] Half-Life 2 (Digital Game). Valve Corporation Seattle, WA,2004.

[2] Appelman, R. Experiential Modes of Game Play. In Situated Play, Proceedings of the DiGRA 2007 Conference (Tokyo,Japan). DiGRA, 2007, 815-822.

[3] Brown, E. and Cairns, P. A Grounded Investigation of Game Immersion. In Conference on Human Factors in Computing Systems (Vienna, Austria). ACM Press, 2004. 1297-1300.

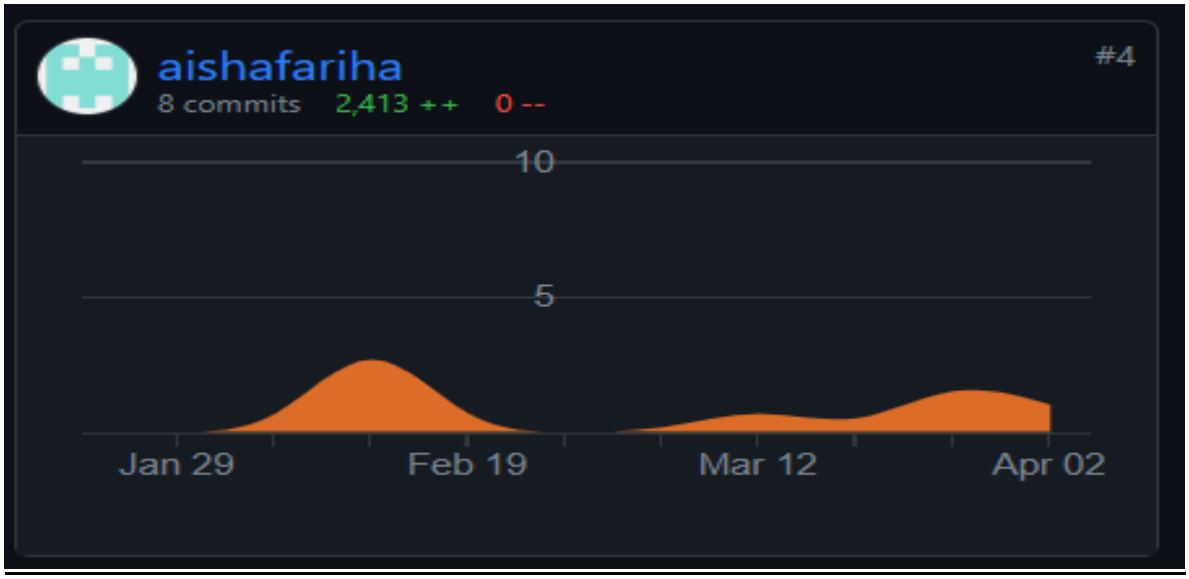
[4] VVorderer, P., Wirth, W., Gouveia, F. R., Biocca, F., Saari, T., Jäncke, F., Böcking, S., Schramm, H., Gysbers, A., Hartmann, T., Klimmt, C., Laarni, J., Ravaja, N., Sacau, A., Baumgartner, T. and Jäncke, P. Mec Spatial Presence Questionnaire (Mecspq): Short Documentation and Instructions for Application. IST-2001-37661, 2004. Report to the European Community, Project Presence: MEC (IST2001-37661).

Link:https://www.youtube.com/watch?v=jO6qQDNa2UY

GitHub & Task Scheduler

For uploading this game all file we are using github and for task scheduler we are using trello.

https://github.com/aishafariha/software



https://github.com/fahimabrar256

https://trello.com/u/ayeshasiddikafariha/boards

https://trello.com/b/GH0ALeuX/player-create