Sabir Yusuf, G4A

# Spider Invasion - Erweiterung

Dies ist ein altes Projekt aus dem Ergänzungsfach Informatik, welches nun als Abschlussprojekt erweitert wurde. Zu den Neuerungen gehört vor allem das Spiel in der Fullscreen mit einem Startmenü, in der der neue Multiplayermodus ausgewählt werden kann. Neben der neuen Co-Op Funktion, bei der ein Drache gesteuert wird, wurden ebenfalls «Power-Up's» hinzugefügt, welche die Lebenspunkte wieder auffüllen oder die Bewegungsgeschwindigkeit temporär steigern. Ausserdem wurde das Spielerlebnis durch diverse Anpassungen verbessert und der Code optimiert.

## Notwendigkeiten

Um dieses Projekt zu starten sind keine speziellen Anforderungen notwendig. Alle nötigen Unterlagen befinden sich in der Zip-Datei.

## Anforderungen

**Das Spiel ist in der Fullscreen und hat einem Startbildschirm mit den Knöpfen «Singleplayer» und «Multiplayer».**


Es wurde zusätzlich ein Informationsknopf in der Ecke eingefügt, um die Steuerung, sowie die Fähigkeiten der jeweiligen Spieler zu erklären.

<u>Fullscreen</u>

```
display_size = pygame.display.get_desktop_sizes()
SCREEN_WIDTH = display_size[0][0]
SCREEN_HEIGHT = display_size[0][1]
```

Der Fullscreen war nicht besonders kompliziert zu implementieren. Der dazu nötige Befehl ergab sich aus einer Internetrecherche. (siehe Quellen)

Startbildschirm mit den Knöpfen «Singleplayer» und «Multiplayer»

```python
import pygame

#button class
class Button():
    def __init__(self, x, y, image, scale):
        width = image.get_width()
        height = image.get_height()
        self.image = pygame.transform.scale(image, (int(width * scale), int(height * scale)))
        self.rect = self.image.get_rect()
        self.rect.topleft = (x, y)
        self.clicked = False

    def draw(self, surface):
        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:
                self.clicked = True
                action = True

        if pygame.mouse.get_pressed()[0] == 0:
            self.clicked = False

        #draw button on screen
        surface.blit(self.image, (self.rect.x, self.rect.y))

        return action
```

Bei der Recherche wurde die Entscheidung getroffen, den Code für den Knopf aus dem Internet zu holen aufgrund mangelnden Wissens. Dieser Code bildet die Basis für die Knöpfe, welche schlussendlich im Spiel integriert werden sollen. (siehe Quellen)

```python
#Button Images
sp_img = pygame.image.load("assets/buttons/singleplayer.png").convert_alpha()
sp_scale = pygame.transform.scale(sp_img, ((SCREEN_WIDTH/2), SCREEN_HEIGHT/3))

mp_img = pygame.image.load("assets/buttons/multiplayer.png").convert_alpha()
mp_scale = pygame.transform.scale(mp_img, ((SCREEN_WIDTH/2), SCREEN_HEIGHT/3))
```

Zuerst wurden die Bilder der Knöpfe im Code eingefügt und entsprechend skaliert. Diese Knöpfe wurden selbstständig in Photoshop erstellt, nur die Rahmen sind aus dem Internet. (siehe Quellen)

```python
#Button creation
sp_button = button.Button(screen.get_rect().centerx-sp_scale.get_rect().width/2, SCREEN_HEIGHT/4.4, sp_scale, 1)
mp_button = button.Button(screen.get_rect().centerx-sp_scale.get_rect().width/2, SCREEN_HEIGHT/2.1, mp_scale, 1)
replay_button = button.Button(screen.get_rect().centerx-sp_scale.get_rect().width/2, SCREEN_HEIGHT/1.5, replay_scale, 1)
info_button = button.Button(SCREEN_WIDTH - 150, SCREEN_HEIGHT - 150, info_scale, 1)
back_button = button.Button(SCREEN_WIDTH - 150, SCREEN_HEIGHT - 150, back_scale, 1)
```

Anschliessend mussten die Knöpfe erstellt werden, dabei wird ebenfalls ihre Position auf dem Display festgelegt. Wichtig ist zu beachten, dass der Basiscode, welcher eine separate .py Dabei ist, in dieses Projekt importiert werden musste, um genutzt werden zu können.

```python
#Draw Pausescreen
def draw_pause():
    screen.fill((0, 0, 0))
    screen.blit(background_dark, (0,0))
    screen.blit(menu, menu.get_rect(center=screen.get_rect().center))
    sp_button.draw(screen)
    mp_button.draw(screen)
    info_button.draw(screen)
```

Schliesslich konnten die Knöpfe auf dem Bildschirm gezeichnet werden. Bisher kann der Knopf zwar gedruckt werden, jedoch wurde noch nicht programmiert, welche Aktion der Knopf ausführen soll.

```python
#Pausescreen
if game_paused == True and game_finished == True and game_info == False:
    if sp_button.draw(screen):
        game_paused = False
        game_finished = False
        gamemode = "singleplayer"
        player = PlayerOne(SCREEN_HEIGHT/2, SCREEN_WIDTH/2)

    if mp_button.draw(screen):
        game_paused = False
        game_finished = False
        gamemode = "multiplayer"
```

Mit einem if-Befehl kann man durch das Klicken auf den Knopf festlegen, was dessen Funktion sein soll. Im Beispiel wird nach dem Knopfdruck der Spielmodus eingestellt und der Pausescreen deaktiviert. Somit erhält der Code den Befehl, das Spiel entsprechend den ausgewählten Einstellungen zu starten.

**Co-Op Möglichkeit, zweiter Spieler mit anderen Fähigkeiten als der erste Spieler.**

<u>Zweiter Spieler</u>

Da bei vielen Co-Op Modi der Mitspieler in der Regel weniger Spielerfahrung als der erste Spieler aufweist, sollte dieser stärker als der Primärspieler sein. Somit wurde sich bei dem zweiten Spieler für einen Drachen, «The Guardian», entschieden. Dieser hat eine unterstützende Rolle und profitiert davon, unverwundbar zu sein. Weitere Unterschiede in seinen Fähigkeiten im Vergleich zum primären Spieler sind eine höhere Bewegungsgeschwindigkeit, die unendliche Anzahl an Feuerbällen und dessen Flammen.

```
#Players
class PlayerOne:···

class PlayerTwo:
    def __init__(self, x, y):···

    #Playermovement & orientation
    def move_player(self, input):···

    #Draw Player
    def draw(self, screen):···

    #Orientation creation
    def direction(self):···

    #Weaponcooldown
    def cooldown(self):···

    #Shooting
    def shoot(self):···
```

Der zweite Spieler ist in der Programmierung ähnlich zum ersten Spieler, wobei sehr viele Variablen geändert wurden. Ursprünglich war der Plan, den Code des ersten Spielers zu nutzen und für den zweiten Spieler nur ein neues Objekt herzustellen. Jedoch erwies sich, dass der zweite Spieler so viele Variablen besitzt, dass es angenehmer war, für diesen eine separate Klasse zu erstellen.

```
#Flame
class Flame(pygame.sprite.Sprite):
    def __init__(self,x,y):
        pygame.sprite.Sprite.__init__(self)
        self.img = pygame.image.load("assets/weapons/flame.png").convert()
        self.img.set_colorkey((255,255,255), RLEACCEL)
        self.flame_scaled = pygame.transform.scale(self.img, (50,70))
        self.rect = self.flame_scaled.get_rect()

        self.rect.x = x
        self.rect.y = y

    #Random Flames after Fireball
    def random_flame(self):
        for fireball in fireballs_group:
            i = random.randint(1,flame_chance)
            if i == 1:
                flame = Flame(fireball.rect.x, fireball.rect.y)
                player2.flames.append(flame)
                flames_group.add(flame)
```

Ein Zusatz beim zweiten Spieler ist die Flamme, welche sich bei den Feuerbällen bilden kann. Diese setzen den Boden in Brand und bleiben so lange, bis ein Gegner mit ihnen kollidiert und anschliessend stirbt.

<u>Co-Op Möglichkeit</u>

```
if gamemode == "singleplayer":
    player = PlayerOne(SCREEN_HEIGHT/2, SCREEN_WIDTH/2)
if gamemode == "multiplayer":
    player = PlayerOne(SCREEN_HEIGHT/2, SCREEN_WIDTH/3)
    player2 = PlayerTwo(SCREEN_HEIGHT/2, SCREEN_WIDTH/3*2)
```

Nach dem Knopfdruck im Startmenü wird der «gamemode» festgesetzt und der Drache wird nur erstellt, wenn der Multiplayermodus ausgewählt wurde. Entsprechend haben die Spieler unterschiedliche Startpositionen.

**Zufällig erscheinen Items für Gesundheitswiederherstellung und mehr Geschwindigkeit**.

OOP bei den Power-Up's

```python
#Power-Ups
class Powerup(pygame.sprite.Sprite):
    def __init__(self,x,y,powerup):
        pygame.sprite.Sprite.__init__(self)
        self.img = pygame.image.load("assets/powerups/" + powerup + ".png").convert()
        self.img.set_colorkey((255,255,255), RLEACCEL)
        self.img_scaled = pygame.transform.scale(self.img, (50,50))
        self.rect = self.img_scaled.get_rect()

        self.rect.x = x
        self.rect.y = y
```

Die Klasse «Powerup» konnte gleich zweimal benutz werden, für die Gesundheitswiederherstellung und die Geschwindigkeitssteigerung. In diesem Fall wurde die Variable «powerup» genutzt, um die jeweiligen Bilder der Power-Up's in die Klasse zu laden.

```python
speed = Powerup(x,y,"speed")
heal = Powerup(x,y,"heal")
```

Somit konnten zwei Objekte aus derselben Klasse erzeugt werden. Diese Methode sollte ursprünglich bei den beiden Spielern ebenfalls genutzt werden, wurde jedoch aufgrund zu vielen Variablen als sehr umständlich angesehen.

```python
#Spawn Power-Ups
i = random.randint(1, powerup_chance)
if i == 1:
    i = random.randint(1, 2)
    if i == 1:
        speed = Powerup(random.randint(0,SCREEN_WIDTH), random.randint(0,(SCREEN_HEIGHT-50)), "speed")
        speed_group.add(speed)

    if i == 2:
        heal = Powerup(random.randint(0,SCREEN_WIDTH), random.randint(0,(SCREEN_HEIGHT-50)), "heal")
        heal_group.add(heal)
```

Nach jedem getöteten Gegner sollte der Spieler die Chance auf einen Power-Up haben. Hierfür wird mit jedem Kill die Variable «i» neu ausgewählt. Erhält diese Variable den Wert 1 oder 2, so wird der jeweilige Power-Up irgendwo auf dem Spielfeld zufällig erstellt. Die Variable «powerup_chance» kann angepasst werden, um die Wahrscheinlichkeit der Power-Up's zu ändern.

```python
#Collision PlayerOne/PowerUp
speed_collision = pygame.sprite.spritecollide(player, speed_group, True)
heal_collision = pygame.sprite.spritecollide(player, heal_group, True)
```

Damit der Spieler mit den Power-Up's interagieren kann, muss die Kollision programmiert werden. Die Power-Up's müssen in separaten Gruppen sein, um dem Spieler bei den Kollisionen den richtigen Effekt geben zu können.

```
#PowerUp (Speed)
if speed_collision:
    player.pace = player1_pace + speed_powerup
    speed_duration = speedboost_duration

if player.pace == player1_pace + speed_powerup:
    speed_duration -= 1
    if speed_duration <= 0:
        player.pace = player1_pace
```

Schliesslich wird nach der Kollision der entsprechende Effekt auf den Spieler übertragen. Die Geschwindigkeitsänderung ist so programmiert, dass die Geschwindigkeitszunahme, sowie der Dauer des Effekts unter den «Game Settings» im Code angepasst werden können.

# Reflexion

## Übersicht

Bei dieser Arbeit habe ich mich an meinem alten Code bedient und war, obwohl ich diesen am besten kennen sollte, manchmal überfragt, da nicht übersichtlich programmiert wurde und einiges an Code nicht klar durch einen Kommentar gekennzeichnet war. Aus diesem Grund musste ich vieles vom Code neu studieren, was durch übersichtlicheres Programmieren hätte verhindert werden können. Diese Übersicht habe ich in dieser Version so gut ich kann hergestellt. Somit ist meine wichtigste Erkenntnis aus dieser Arbeit, folgende Fehler nicht mehr zu machen:

Variablen

```
#Game Settings

# 1 = lowest
player1_pace = 5
player2_pace = 7
spider_pace = 3

speed_powerup = 2

# 1s = 100
speedboost_duration = 300

# 1 = fastest
player1_cooldown_duration = 20
player2_cooldown_duration = 30

# 1 = most frequent
flame_chance= 4
powerup_chance=8

# 1 HP = 100
heal_boost = 1000

# 1 = least
amount_spiders=1
```

An diversen Stellen im Code hatte ich sehr grosse Mühe dabei, den Code anzupassen, da die kleinste Veränderung sehr viele Fehlermeldungen zur Folge hatte und mühselige Anpassungsarbeit erforderte. Ich habe viel Zeit darin investiert, vieles durch Variablen zu ersetzen, um dieses Problem zu minimieren. Ausserdem habe ich viele Konstanten zu Variablen gemacht und somit «Game Settings» ermöglicht.

Einheitlichkeit

```python
#Flame
class Flame(pygame.sprite.Sprite):
    def __init__(self,x,y):
        pygame.sprite.Sprite.__init__(self)
        self.img = pygame.image.load("assets/weapons/flame.png").convert()
        self.img.set_colorkey((255,255,255), RLEACCEL)
        self.img_scaled = pygame.transform.scale(self.img, (50,70))
        self.rect = self.img_scaled.get_rect()

        self.rect.x = x
        self.rect.y = y
```

Manche Variablen waren auf Deutsch, manche auf Englisch, dies sorgte bei mir manchmal für Verwirrung und sah nicht besonders schön aus. Diese wurden alle auf Englisch übersetzt und, wenn angebracht, verständlicher gewählt.

Beispiel:

Die Klassen sind identisch aufgebaut, dabei gab es unnötige Unterschiede in den Namen der Variablen. Diese wurden auf Einheitlichkeit optimiert und haben dieselben Variablennamen. Diese Einheitlichkeit verbesserte die Übersicht enorm.

## Knöpfe

Ich habe gelernt, wie man mit Knöpfen arbeitet, was bei jeglicher Software von immenser Bedeutung ist, um die Kommunikation zwischen Menschen und Maschinen zu ermöglichen.

```python
import pygame

#button class
class Button():
    def __init__(self, x, y, image, scale):
        width = image.get_width()
        height = image.get_height()
        self.image = pygame.transform.scale(image, (int(width * scale), int(height * scale)))
        self.rect = self.image.get_rect()
        self.rect.topleft = (x, y)
        self.clicked = False

    def draw(self, surface):
        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:
                self.clicked = True
                action = True

        if pygame.mouse.get_pressed()[0] == 0:
            self.clicked = False

        #draw button on screen
        surface.blit(self.image, (self.rect.x, self.rect.y))

        return action
```

# Quellen von Programmcode

Fullscreen

https://www.pygame.org/docs/ref/display.html#pygame.display.get_desktop_sizes

Buttons

https://github.com/russs123/pygame_tutorials/blob/main/Menu/button.py

Rahmen

https://de.vecteezy.com/vektorkunst/6203095-set-mittelalter-metallic-frame-overlay-game