# 吸血鬼生存猪

## 実行環境の必要条件

- python >= 3.10
- pygame >= 2.1

## ゲームの概要

- 主人公キャラクターをマウス操作により敵をにするゲームで，．．．
- 参考URL：サイトタイトル

## ゲームの遊び方

- 次々と現れる敵を避けながら戦い、強力な攻撃アイテムや特殊なスキルを獲得してキャラクターを強化
- 最終的に画面を埋め尽くすほどの敵を倒していく

## ゲームの実装

### 共通基本機能

- 背景画像と主人公キャラクターの描画
- プレイヤーのレベルアップ時に、強化するスキルを選択する画面表示

### 分担追加機能

- ステージ移動：移動方向とマウスの座標取得でキャラクターの移動
- 攻撃パターン：全キャラクターの攻撃方法とターゲットの決定
- スキル画面：どんなスキルを追加するのか、マウス押下でスキル選択できるUI画面
- レベルアップ：敵を倒した時に、その敵の種類に応じて経験値を決定

### ToDo

- [ ]
- [ ]

## メモ

- クラス内の変数は，すべて「get_変数名」という名前のメソッドを介してアクセスするように設計してある
- すべてのクラスに関係する関数は，クラスの外で定義してある

```python
import os
import sys
import pygame as pg
import random
import math

# ゲームの初期設定
WIDTH, HEIGHT = 1800, 1000
os.chdir(os.path.dirname(os.path.abspath(__file__)))

# ゲームキャラクター（主人公、敵、弾、経験値）に関するクラス
class Bird(pg.sprite.Sprite):
    def __init__(self, num: int, xy: tuple[int, int]):
        """
        引数１ num:こうかとん画像ファイル名の番号
        引数２ xy:こうかとん画像の位置座標タプル
        """
        super().__init__()
        img = pg.image.load(f"fig/{num}.png")
        img0 = pg.transform.flip(img, True, False)
        self.original_image = pg.transform.scale(img, (60, 60))
        self.image = self.original_image
        self.rect = self.image.get_rect(center=xy)
        self.speed = 5
        self.exp = 0
        self.hp = 50  # 主人公のHP
        self.imgs = {
            (1,  0): img0,  # 右
            (1, -1): pg.transform.rotozoom(img0, 45, 1.0),  # 右上
            (0, -1): pg.transform.rotozoom(img0, 90, 1.0),  # 上
            (-1, -1): pg.transform.rotozoom(img, -45, 1.0),  # 左上
            (-1,  0): img,  # 左
            (-1,  1): pg.transform.rotozoom(img, 45, 1.0),  # 左下
            (0,  1): pg.transform.rotozoom(img0, -90, 1.0),  # 下
            (1,  1): pg.transform.rotozoom(img0, -45, 1.0),  # 右下
        }
```

```python
        self.image = self.imgs[(1, 0)] # 初期画像


    def update(self, mousu_pos):
        # dx, dy = mousu_pos[0] - self.rect.centerx, mousu_pos[1] - self.rect.centery
        # distance = math.hypot(dx, dy)
        # if distance > 0:
        #     dx, dy = dx / distance, dy / distance
        #     self.rect.centerx += dx * self.speed
        #     self.rect.centery += dy * self.speed
        # マウスの方向に移動
        mousu_pos = pg.mouse.get_pos()
        dx, dy = mousu_pos[0] - self.rect.centerx, mousu_pos[1] - self.rect.centery
        distance = math.hypot(dx, dy)
        if distance > 0:
            dx, dy = dx / distance, dy / distance
            self.rect.centerx += dx * 5  # スピード調整
            self.rect.centery += dy * 5

        angle = math.degrees(math.atan2(-dy, dx)) # Y軸反転 → 角度計算

        if -22.5 <= angle < 22.5:
            direction = (1, 0)          #右
        elif 22.5 <= angle < 67.5:
            direction = (1, -1)         #右上
        elif 67.5 <= angle < 112.5:
            direction = (0, -1)         #上
        elif 112.5 <= angle < 157.5:
            direction = (-1, -1)        #左上
        elif 157.5 <= angle or angle < -157.5:
            direction = (-1, 0)         #左
        elif -157.5 <= angle < -112.5:
            direction = (-1, 1)         #左下
        elif -112.5 <= angle < -67.5:
            direction = (0, 1)          #下
        elif -67.5 <= angle < -22.5:
            direction = (1, 1)          #右下

        self.image = self.imgs[direction]
        self.rect = self.image.get_rect(center = self.rect.center) # 画像の中心座標を維持

    def gain_exp(self, value):
        self.exp += value
        print(f"経験値: {self.exp}")

    def take_damage(self):
        # 敵と接近した時にダメージを受ける処理
        self.hp -= 1
        print(f"hp:{self.hp}")
        if self.hp <= 0:
            print("Game Over")
            self.kill()
```

```python
class Enemy(pg.sprite.Sprite):
    def __init__(self, xy: tuple[int, int]):
        super().__init__()
        self.image = pg.Surface((40, 40), pg.SRCALPHA)
        pg.draw.circle(self.image, (255, 0, 0), (20, 20), 20)
        self.rect = self.image.get_rect(center=xy)
        self.speed = 2
        self.health = 3

    def update(self, target):
        dx, dy = target[0] - self.rect.centerx, target[1] - self.rect.centery
        distance = math.hypot(dx, dy)
        if distance > 0:
            dx, dy = dx / distance, dy / distance
            self.rect.centerx += dx * self.speed
            self.rect.centery += dy * self.speed

    def hit(self):
        self.health -= 1
        if self.health <= 0:
            self.kill()
            return ExpOrb(self.rect.center)
        return None

class Bullet(pg.sprite.Sprite):
    def __init__(self, pos, target_pos):
        super().__init__()
        self.image = pg.Surface((10, 10), pg.SRCALPHA)
        pg.draw.circle(self.image, (0, 255, 255), (5, 5), 5)
        self.rect = self.image.get_rect(center=pos) # 弾の初期位置を設定
        # 弾の移動速度と方向の計算
        dx, dy = target_pos[0] - pos[0], target_pos[1] - pos[1] # ターゲットへの距離
        distance = math.hypot(dx, dy) # ターゲットまでの直線距離（ユークリッド距離）
        self.speed = 10
        # 正規化して速度ベクトルを求める
        self.velocity = (dx / distance * self.speed, dy / distance * self.speed)

    def update(self):
        # 弾の位置更新（速度ベクトルに従って直進）
        self.rect.x += self.velocity[0] #0:x方向の移動量を設定
        self.rect.y += self.velocity[1] #1:y方向の移動量を設定
        # 画面外に出た弾を削除
        if not (0 <= self.rect.x <= WIDTH and 0 <= self.rect.y <= HEIGHT):
            self.kill() #spriteグループから削除

class ExpOrb(pg.sprite.Sprite):
    def __init__(self, pos):
        super().__init__()
        self.image = pg.Surface((15, 15), pg.SRCALPHA)
        pg.draw.circle(self.image, (0, 255, 0), (7, 7), 7)
        self.rect = self.image.get_rect(center=pos)
        self.value = 10
```

```python

#敵の再出現を管理するクラス
class EnemyManager:
    def __init__(self, all_sprites, enemies, respawn_time=300):
        self.all_sprites = all_sprites
        self.enemies = enemies
        self.respawn_time = respawn_time #  復活するまでのフレーム数
        self.respawn_timer = []

    def update(self):
        #敵の再出現を管理
        for idx, timer in enumerate(self.respawn_timer):
            self.respawn_timer[idx] -= 1
            if self.respawn_timer[idx] <= 0:
                self.respawn_timer.pop(idx)
                self.spawn_enemy()
        #敵の数を5～7体に保つ
        if len(self.enemies) < 5:
            self.spawn_enemy()

    def spawn_enemy(self):
        enemy = Enemy((random.randint(0, WIDTH), random.randint(0, HEIGHT)))
        self.enemies.add(enemy)
        self.all_sprites.add(enemy)

    def schedule_respawn(self):
        if len(self.enemies) < 7:
            self.respawn_timer.append(self.respawn_time)

# ゲームのメインループ
def main():
    pg.display.set_caption("吸血鬼生存猪")
    screen = pg.display.set_mode((WIDTH, HEIGHT))
    clock = pg.time.Clock()
    font = pg.font.Font(None, 36)

    # プレイヤーと敵の初期化
    bird = Bird(3, (WIDTH // 2, HEIGHT // 2))
    enemies = pg.sprite.Group(Enemy((random.randint(0, WIDTH), random.randint(0, HEIGHT)
    bullets = pg.sprite.Group()
    exp_orbs = pg.sprite.Group()

    all_sprites = pg.sprite.Group(bird, *enemies)
    enemy_manager = EnemyManager(all_sprites, enemies)

    bullet_timer = 0
    game_over = False  # ゲームオーバー状態のフラグ
    game_over_time = 0  # ゲームオーバー時刻


    while True:
        for event in pg.event.get():
```

```
            if event.type == pg.QUIT:
                return

        if not game_over:
            # 1番近い敵を探す
            if enemies:
                closest_enemy = min(enemies, key=lambda e: math.hypot(e.rect.centerx - b
                if bullet_timer <= 0:
                    bullet = Bullet(bird.rect.center, closest_enemy.rect.center)
                    bullets.add(bullet)
                    all_sprites.add(bullet)
                    bullet_timer = 30  # 弾発射の間隔フレーム

            # 弾と敵の衝突判定
            for bullet in pg.sprite.groupcollide(bullets, enemies, True, False).keys():
                orb = closest_enemy.hit()
                if orb:
                    exp_orbs.add(orb)
                    all_sprites.add(orb)
                    enemy_manager.schedule_respawn()


            # 経験値玉と主人公の衝突判定
            for orb in pg.sprite.spritecollide(bird, exp_orbs, True):
                bird.gain_exp(orb.value)

            # 主人公が敵に接触した時ときのダメージ
            if pg.sprite.spritecollide(bird, enemies, False):
                bird.take_damage()
                if bird.hp <= 0:
                    game_over = True
                    game_over_time = pg.time.get_ticks()  # ゲームオーバー時刻を記録

            # 更新処理
            mouse_pos = pg.mouse.get_pos()
            bird.update(mouse_pos)
            enemies.update(bird.rect.center)
            bullets.update()
            enemy_manager.update()

            bullet_timer -= 1

        # 画面更新
        screen.fill((30, 30, 30))
        all_sprites.draw(screen)

        # HPと経験値の表示
        hp_text = font.render(f"HP: {bird.hp}", True, (255, 255, 255))
        exp_text = font.render(f"EXP: {bird.exp}", True, (255, 255, 255))
        screen.blit(hp_text, (10, 10))
        screen.blit(exp_text, (10, 50))

        #ゲームオーバー画面の処理
```

```
247             if game_over:
248                 game_over_text = font.render("Game Over", True, (255, 0, 0))
249                 screen.blit(game_over_text, (WIDTH // 2 - game_over_text.get_width() // 2, H
250                 if pg.time.get_ticks() - game_over_time > 2000:  # 2秒経過したらゲーム終了
251                     pg.quit()
252                     sys.exit()
253 
254             pg.display.update()
255             clock.tick(60)
256 
257     if __name__ == "__main__":
258         pg.init()
259         main()
260         pg.quit()
261         sys.exit()
```