



31 lines (25 loc) · 1.38 KB

Preview

Code

Blame

Raw



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する関数は、クラスの外で定義してある

c0b2306162 /
ProjExD_Group0X

<> Code

Issues

Pull requests

Actions

Projects

Wiki

Security



C0B23061/Signifi... ▾

ProjExD_Group0X / kokaton_surv.py



c0b2306162 各判定完成

bf572fa · 10 hours ago



261 lines (224 loc) · 9.66 KB

Code

Blame

Raw



```
1  import os
2  import sys
3  import pygame as pg
4  import random
5  import math
6
7  # ゲームの初期設定
8  WIDTH, HEIGHT = 1800, 1000
9  os.chdir(os.path.dirname(os.path.abspath(__file__)))
10
11 # ゲームキャラクター（主人公、敵、弾、経験値）に関するクラス
12 class Bird(pg.sprite.Sprite):
13     def __init__(self, num: int, xy: tuple[int, int]):
14         """
15         引数1 num:こうかとん画像ファイル名の番号
16         引数2 xy:こうかとん画像の位置座標タプル
17         """
18         super().__init__()
19         img = pg.image.load(f"fig/{num}.png")
20         img0 = pg.transform.flip(img, True, False)
21         self.original_image = pg.transform.scale(img, (60, 60))
22         self.image = self.original_image
23         self.rect = self.image.get_rect(center=xy)
24         self.speed = 5
25         self.exp = 0
26         self.hp = 50 # 主人公のHP
27         self.imgs = {
28             (1, 0): img0, # 右
29             (1, -1): pg.transform.rotozoom(img0, 45, 1.0), # 右上
30             (0, -1): pg.transform.rotozoom(img0, 90, 1.0), # 上
31             (-1, -1): pg.transform.rotozoom(img, -45, 1.0), # 左上
32             (-1, 0): img, # 左
33             (-1, 1): pg.transform.rotozoom(img, 45, 1.0), # 左下
34             (0, 1): pg.transform.rotozoom(img0, -90, 1.0), # 下
35             (1, 1): pg.transform.rotozoom(img0, 135, 1.0), # 右下
36         }
```

```

37     self.image = self.imgs[(1, 0)] # 初期画像
38
39
40     def update(self, mousu_pos):
41         # dx, dy = mousu_pos[0] - self.rect.centerx, mousu_pos[1] - self.rect.centery
42         # distance = math.hypot(dx, dy)
43         # if distance > 0:
44         #     dx, dy = dx / distance, dy / distance
45         #     self.rect.centerx += dx * self.speed
46         #     self.rect.centery += dy * self.speed
47         # マウスの方向に移動
48         mousu_pos = pg.mouse.get_pos()
49         dx, dy = mousu_pos[0] - self.rect.centerx, mousu_pos[1] - self.rect.centery
50         distance = math.hypot(dx, dy)
51         if distance > 0:
52             dx, dy = dx / distance, dy / distance
53             self.rect.centerx += dx * 5 # スピード調整
54             self.rect.centery += dy * 5
55
56         angle = math.degrees(math.atan2(-dy, dx)) # Y軸反転 → 角度計算
57
58         if -22.5 <= angle < 22.5:
59             direction = (1, 0) # 右
60         elif 22.5 <= angle < 67.5:
61             direction = (1, -1) # 右上
62         elif 67.5 <= angle < 112.5:
63             direction = (0, -1) # 上
64         elif 112.5 <= angle < 157.5:
65             direction = (-1, -1) # 左上
66         elif 157.5 <= angle or angle < -157.5:
67             direction = (-1, 0) # 左
68         elif -157.5 <= angle < -112.5:
69             direction = (-1, 1) # 左下
70         elif -112.5 <= angle < -67.5:
71             direction = (0, 1) # 下
72         elif -67.5 <= angle < -22.5:
73             direction = (1, 1) # 右下
74
75         self.image = self.imgs[direction]
76         self.rect = self.image.get_rect(center = self.rect.center) # 画像の中心座標を維持
77
78     def gain_exp(self, value):
79         self.exp += value
80         print(f"経験値: {self.exp}")
81
82     def take_damage(self):
83         # 敵と接近した時にダメージを受ける処理
84         self.hp -= 1
85         print(f"hp:{self.hp}")
86         if self.hp <= 0:
87             print("Game Over")
88             self.kill()
89

```

```
90  ✓ class Enemy(pg.sprite.Sprite):
91  ✓     def __init__(self, xy: tuple[int, int]):
92         super().__init__()
93         self.image = pg.Surface((40, 40), pg.SRCALPHA)
94         pg.draw.circle(self.image, (255, 0, 0), (20, 20), 20)
95         self.rect = self.image.get_rect(center=xy)
96         self.speed = 2
97         self.health = 3
98
99  ✓     def update(self, target):
100         dx, dy = target[0] - self.rect.centerx, target[1] - self.rect.centery
101         distance = math.hypot(dx, dy)
102         if distance > 0:
103             dx, dy = dx / distance, dy / distance
104             self.rect.centerx += dx * self.speed
105             self.rect.centery += dy * self.speed
106
107  ✓     def hit(self):
108         self.health -= 1
109         if self.health <= 0:
110             self.kill()
111             return ExpOrb(self.rect.center)
112         return None
113
114  ✓ class Bullet(pg.sprite.Sprite):
115  ✓     def __init__(self, pos, target_pos):
116         super().__init__()
117         self.image = pg.Surface((10, 10), pg.SRCALPHA)
118         pg.draw.circle(self.image, (0, 255, 255), (5, 5), 5)
119         self.rect = self.image.get_rect(center=pos) # 弾の初期位置を設定
120         # 弾の移動速度と方向の計算
121         dx, dy = target_pos[0] - pos[0], target_pos[1] - pos[1] # ターゲットへの距離
122         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)
123         self.speed = 10
124         # 正規化して速度ベクトルを求める
125         self.velocity = (dx / distance * self.speed, dy / distance * self.speed)
126
127  ✓     def update(self):
128         # 弾の位置更新 (速度ベクトルに従って直進)
129         self.rect.x += self.velocity[0] #0:x方向の移動量を設定
130         self.rect.y += self.velocity[1] #1:y方向の移動量を設定
131         # 画面外に出た弾を削除
132         if not (0 <= self.rect.x <= WIDTH and 0 <= self.rect.y <= HEIGHT):
133             self.kill() #spriteグループから削除
134
135  ✓ class ExpOrb(pg.sprite.Sprite):
136  ✓     def __init__(self, pos):
137         super().__init__()
138         self.image = pg.Surface((15, 15), pg.SRCALPHA)
139         pg.draw.circle(self.image, (0, 255, 0), (7, 7), 7)
140         self.rect = self.image.get_rect(center=pos)
141         self.value = 10
```

```
142
143     #敵の再出現を管理するクラス
144     ✓ class EnemyManager:
145     ✓     def __init__(self, all_sprites, enemies, respawn_time=300):
146         self.all_sprites = all_sprites
147         self.enemies = enemies
148         self.respawn_time = respawn_time # 復活するまでのフレーム数
149         self.respawn_timer = []
150
151     ✓     def update(self):
152         #敵の再出現を管理
153         for idx, timer in enumerate(self.respawn_timer):
154             self.respawn_timer[idx] -= 1
155             if self.respawn_timer[idx] <= 0:
156                 self.respawn_timer.pop(idx)
157                 self.spawn_enemy()
158         #敵の数を5〜7体に保つ
159         if len(self.enemies) < 5:
160             self.spawn_enemy()
161
162     def spawn_enemy(self):
163         enemy = Enemy((random.randint(0, WIDTH), random.randint(0, HEIGHT)))
164         self.enemies.add(enemy)
165         self.all_sprites.add(enemy)
166
167     def schedule_respawn(self):
168         if len(self.enemies) < 7:
169             self.respawn_timer.append(self.respawn_time)
170
171     # ゲームのメインループ
172     ✓ def main():
173         pg.display.set_caption("吸血鬼生存猪")
174         screen = pg.display.set_mode((WIDTH, HEIGHT))
175         clock = pg.time.Clock()
176         font = pg.font.Font(None, 36)
177
178         # プレイヤーと敵の初期化
179         bird = Bird(3, (WIDTH // 2, HEIGHT // 2))
180         enemies = pg.sprite.Group(Enemy((random.randint(0, WIDTH), random.randint(0, HEIGHT)))
181         bullets = pg.sprite.Group()
182         exp_orbs = pg.sprite.Group()
183
184         all_sprites = pg.sprite.Group(bird, *enemies)
185         enemy_manager = EnemyManager(all_sprites, enemies)
186
187         bullet_timer = 0
188         game_over = False # ゲームオーバー状態のフラグ
189         game_over_time = 0 # ゲームオーバー時刻
190
191
192         while True:
193             for event in pg.event.get():
```

```
194     if event.type == pg.QUIT:
195         return
196
197     if not game_over:
198         # 1番近い敵を探す
199         if enemies:
200             closest_enemy = min(enemies, key=lambda e: math.hypot(e.rect.centerx - bird.rect.centerx, e.rect.centery - bird.rect.centery))
201             if bullet_timer <= 0:
202                 bullet = Bullet(bird.rect.center, closest_enemy.rect.center)
203                 bullets.add(bullet)
204                 all_sprites.add(bullet)
205                 bullet_timer = 30 # 弾発射の間隔フレーム
206
207         # 弾と敵の衝突判定
208         for bullet in pg.sprite.groupcollide(bullets, enemies, True, False).keys():
209             orb = closest_enemy.hit()
210             if orb:
211                 exp_orbs.add(orb)
212                 all_sprites.add(orb)
213                 enemy_manager.schedule_respawn()
214
215         # 経験値玉と主人公の衝突判定
216         for orb in pg.sprite.spritecollide(bird, exp_orbs, True):
217             bird.gain_exp(orb.value)
218
219         # 主人公が敵に接触したときのダメージ
220         if pg.sprite.spritecollide(bird, enemies, False):
221             bird.take_damage()
222             if bird.hp <= 0:
223                 game_over = True
224                 game_over_time = pg.time.get_ticks() # ゲームオーバー時刻を記録
225
226         # 更新処理
227         mouse_pos = pg.mouse.get_pos()
228         bird.update(mouse_pos)
229         enemies.update(bird.rect.center)
230         bullets.update()
231         enemy_manager.update()
232
233         bullet_timer -= 1
234
235         # 画面更新
236         screen.fill((30, 30, 30))
237         all_sprites.draw(screen)
238
239         # HPと経験値の表示
240         hp_text = font.render(f"HP: {bird.hp}", True, (255, 255, 255))
241         exp_text = font.render(f"EXP: {bird.exp}", True, (255, 255, 255))
242         screen.blit(hp_text, (10, 10))
243         screen.blit(exp_text, (10, 50))
244
245         # 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()
```