


 c0b2304920 /
ProjExD_Group08



[Code](#) [Pull requests](#) [Actions](#) [Projects](#) [Wiki](#) [Security](#) [Insights](#) [Settings](#)

[ProjExD_Group08](#) / [README.md](#)  c0b2304920 readme 修正1

14959ca · 10 minutes ago



44 lines (38 loc) · 2.09 KB

[Preview](#) [Code](#) [Blame](#)

[Raw](#)    

ボンバーこうかтон



実行環境の必要条件

- python >= 3.10
- pygame >= 2.1

ゲームの概要

- 障害物も敵も爆弾で破壊するゲーム with こうかといん
- 参考URL : [レトロゲームの殿堂](#)

ゲームの遊び方

- 矢印キーでボンバーKOKAを操作し、スペースキー押下による足元への爆弾設置。
- 三分以内に他キャラクターを殲滅、もしくは最大スコアの状態で勝利。

ゲームの実装

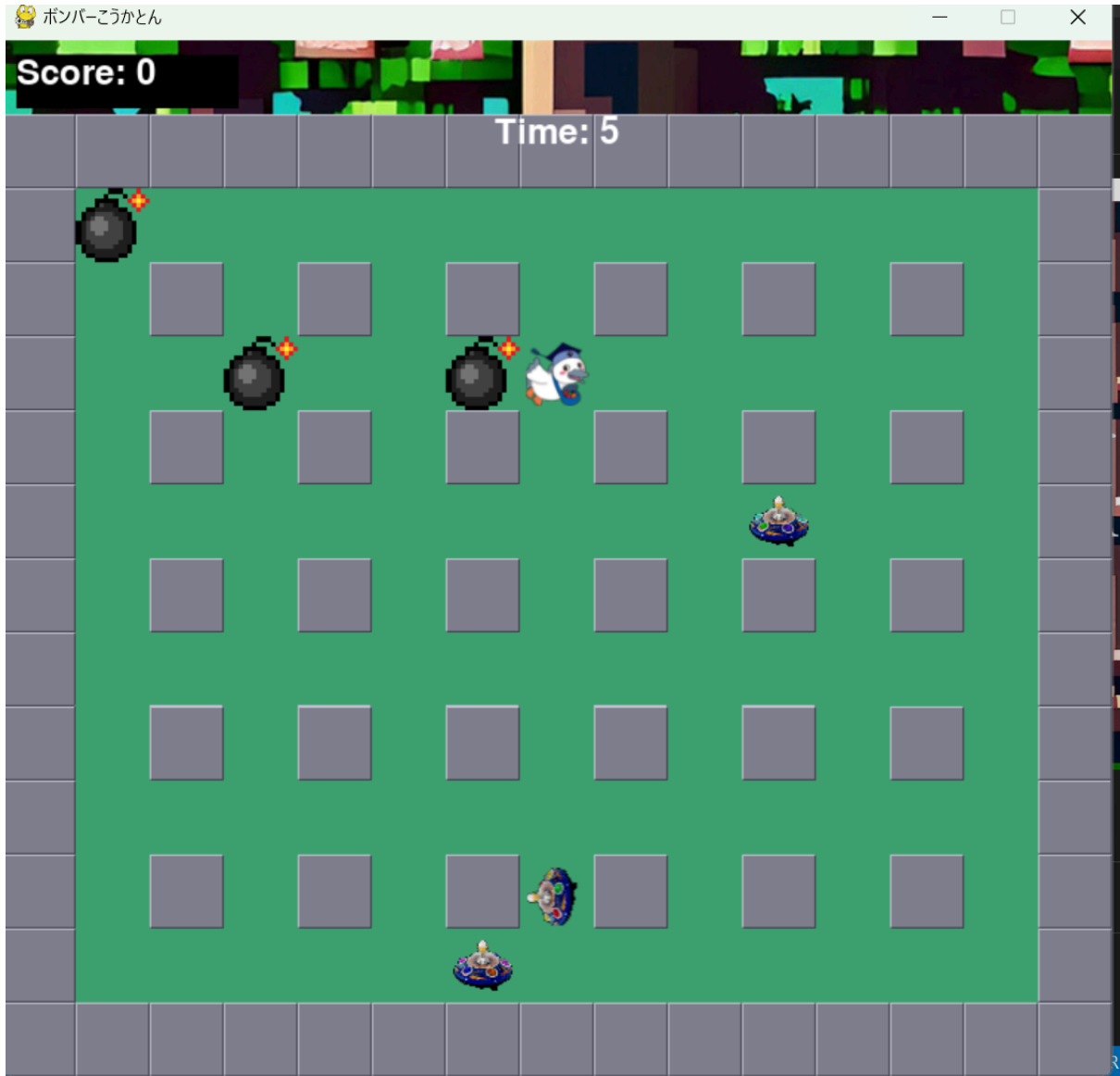
共通基本機能

- 主人公キャラ
- 矢印操作機能
- 盤面領域内判定関数

分担追加機能

- Hero(操作キャラ)クラス(北村) : 操作するキャラクターに関するクラス
- 壁の生成、破壊機能クラス(北村) : エリア内の破壊できない障害物の当たり判定に関する関数
- マップ詳細生成機能クラス(北村) : マップのオブジェクトの生成に関するクラス
- 敵クラス(小林) : 敵を生成するクラス
- 爆弾の制御機能クラス(小林) : 爆弾の設置に関するクラス
- 他キャラの行動機能クラス(小林) : 他キャラクターの動きに関するクラス
- タイマー機能クラス(町田) : 制限時間に関するクラス
- 効果音、BGM制御機能クラス(町田) : 音に関するクラス
- スコア機能クラス(おん) : 敵を倒して時のスコアに関するクラス
- タイトル画面描画機能クラス(岡崎) : ゲームを始めた時のタイトル画面に関するクラス

- ゲームオーバー画面描画クラス(岡崎) : ゲームが終了したときの画面に関するクラス



ToDo

- [1] マージ後のコメントなどの精査、修正
- [2] 記述方法の統一
- [3] 素材作成
- [4] 細かな機能の発案、実装
- [5] アイテム実装

メモ

- main関数は最低限の呼び出しのみで記述している
- 汎用的な関数を用意して、動作対象に適用している

c0b2304920 /
ProjExD_Group08

<> Code

Pull requests

Actions

Projects

Wiki

Security

Insights

Settings

ProjExD_Group08 / bomber_kokaton.py



c0a23059fb ファイルパス、コメント、変数の修正

4f75709 · 18 minutes ago



401 lines (346 loc) · 13.7 KB

Code

Blame

Raw



```
1  import os
2  import random
3  import sys
4  import time
5
6  import pygame as pg
7
8
9  WIDTH, HEIGHT = 750, 700
10 os.chdir(os.path.dirname(os.path.abspath(__file__)))
11
12
13 # スコア表示のクラス
14 class Score:
15     """
16     スコア管理クラス
17     スコアの追跡と更新を処理する
18     """
19     def __init__(self) -> None:
20         self.score = 0 # 初期スコアは0
21
22     def add_score(self, points: int) -> None:
23         self.score += points # スコアを加算
24         print(f"Score: {self.score}") # 現在のスコアを表示（デバッグ用）
25
26     def get_score(self) -> int:
27         return self.score # 現在のスコアを返す
28
29
30 # こうかたん（プレイヤー）のクラス
31 class Hero:
32     """
33     ゲームキャラクター（こうかたん）に関するクラス
34     """
35     delta = {
36         pg.K_UP: (0, -50),
37         pg.K_DOWN: (0, +50),
38         pg.K_LEFT: (-50, 0),
39         pg.K_RIGHT: (+50, 0),
40     }
41     img0 = pg.transform.rotozoom(pg.image.load("images/Kokaton/3.png"), 0, 0.9)
42     img = pg.transform.flip(img0, True, False)
```

```

43  ✓   imgs = {
44       (+50, 0): img,
45       (+50, -50): pg.transform.rotozoom(img, 45, 0.9),
46       (0, -50): pg.transform.rotozoom(img, 90, 0.9),
47       (-50, -50): pg.transform.rotozoom(img0, -45, 0.9),
48       (-50, 0): img0,
49       (-50, +50): pg.transform.rotozoom(img0, 45, 0.9),
50       (0, +50): pg.transform.rotozoom(img, -90, 0.9),
51       (+50, +50): pg.transform.rotozoom(img, -45, 0.9),
52   }
53   mvct = 0
54
55  ✓   def __init__(self, xy: tuple[int, int]) -> None:
56       self.img = __class__.imgs[(+50, 0)]
57       self.rct: pg.Rect = self.img.get_rect()
58       self.rct.center = xy
59       self.dire = (+50, 0)
60       self.score = 0 # スコアの初期化
61
62   def add_score(self, points: int) -> None:
63       """スコアにポイントを追加するメソッド"""
64       self.score += points
65
66   # キーの入力で動く処理
67  ✓   def update(self, key_lst: list[bool], screen: pg.Surface) -> None:
68       """
69       押下キーに応じてこうかとんを移動させる
70       引数1 key_lst: 押下キーの真理値リスト
71       引数2 screen: 画面Surface
72       """
73       sum_mv = [0, 0]
74       if __class__.mvct == 0:
75           for k, mv in __class__.delta.items():
76               if key_lst[k]:
77                   sum_mv[0] += mv[0]
78                   sum_mv[1] += mv[1]
79               if 0 != sum_mv[0] and 0 != sum_mv[1]:
80                   print(8)
81                   sum_mv[0] = 0
82                   sum_mv[1] = 0
83       self.rct.move_ip(sum_mv)
84       __class__.mvct = 15
85       elif 0 < __class__.mvct:
86           __class__.mvct -= 1
87       if check_bound(self.rct) != (True, True):
88           self.rct.move_ip(-sum_mv[0], -sum_mv[1])
89       if not (sum_mv[0] == 0 and sum_mv[1] == 0):
90           self.img = __class__.imgs[tuple(sum_mv)]
91           self.dire = sum_mv
92       screen.blit(self.img, self.rct)
93
94
95   # 敵のクラス
96  ✓   class Enemy(pg.sprite.Sprite):
97       """
98       敵に関するクラス
99       """
100      imgs = [pg.image.load(f"images/ufo/alien{i}.png") for i in range(1, 4)]

```

101

```

102  ✓ def __init__(self, num: int, vx: tuple[int, int]) -> None:
103      """
104      敵のSurfaceの作成
105      引数1 num: 画像指定用整数
106      引数2 vx: Rectのcenter用タプル
107      """
108      super().__init__()
109      self.num = num
110      self.img = __class__.imgs[self.num]
111      self.image = pg.transform.rotozoom(self.img, 0, 0.5)
112      self.rect = self.image.get_rect()
113      self.rect.center = vx
114      self.vx, self.vy = 0, 0
115      self.mvct = 0 # 行動後のクールタイム
116      self.state = "move" # move、bomで行動管理
117
118  ✓ def control(self) -> None:
119      """
120      敵に関する動作制御を行う
121      """
122      img_key = { # 進行方向に応じた画像
123          (+50, 0): pg.transform.rotozoom(self.img, 0, 0.5), # 右
124          (0, -50): pg.transform.rotozoom(self.image, 90, 1.0), # 上
125          (-50, 0): pg.transform.flip(self.image, True, False), # 左
126          (0, +50): pg.transform.rotozoom(self.image, -90, 1.0), # 下
127      }
128      move_list = [ # 移動用数値
129          (0, -50), # 上
130          (0, +50), # 下
131          (-50, 0), # 左
132          (+50, 0), # 右
133      ]
134
135      if self.mvct == 0: # クールタイムでなければ
136          while True:
137              sum_mv = random.choice(move_list)
138              self.rect.move_ip(sum_mv[0], sum_mv[1])
139              if check_bound(self.rect) != (True, True): # 盤面領域判定
140                  self.rect.move_ip(-sum_mv[0], -sum_mv[1])
141                  continue
142                  break
143              self.image = img_key[sum_mv]
144              self.mvct = 15
145      elif self.mvct > 0: # クールタイムであれば
146          self.mvct -= 1
147
148  ✓ def update(self) -> None:
149      """
150      敵の情報を更新する
151      """
152      __class__.control(self)
153
154
155      # 爆弾（ボンバー）のクラス
156  ✓ class Bomber(pg.sprite.Sprite):
157      """
158      爆弾に関するクラス
159      """

```

```
160 ✓ def __init__(self, vx: tuple[int, int], hero: Hero, enemies: pg.sprite.Group) -> None:
161     super().__init__()
162     self.bom_img = pg.image.load("images/bom/bom.png")
163     self.exp_img = pg.image.load("images/bom/explosion.png")
164     self.image = pg.transform.rotozoom(self.bom_img, 0, 0.1)
165     self.rect = self.image.get_rect()
166     self.rect.center = vx
167     self.count = 300
168     self.state = "bom"
169     self.hero = hero
170     self.enemies = enemies
171
172 ✓ def control(self) -> None:
173     """
174     爆弾の動作を処理する
175     """
176     if self.count == 0:
177         if self.state == "bom":
178             self.image = pg.transform.rotozoom(self.exp_img, 180, 0.05)
179             self.count = 30
180             self.state = "explosion"
181         else:
182             # 爆発時に敵と衝突した場合スコアを増加
183             collided_enemies = pg.sprite.spritecollide(self, self.enemies, True)
184             if collided_enemies:
185                 self.hero.add_score(100 * len(collided_enemies))
186                 self.kill()
187             elif self.count > 0:
188                 self.count -= 1
189                 if self.state == "explosion":
190                     self.image = pg.transform.rotate(self.image, 90)
191
192 ✓ def update(self) -> None:
193     """
194     爆弾の情報を更新する
195     """
196     self.control()
197
198
199 # スコア表示クラス
200 ✓ class Score:
201     """
202     スコア管理クラス
203     スコアの追跡と更新を処理する
204     """
205     def __init__(self) -> None:
206         self.score = 0 # 初期スコアは0
207
208     def add_score(self, points: int) -> None:
209         self.score += points # スコアを加算
210         print(f"Score: {self.score}") # 現在のスコアを表示 (デバッグ用)
211
212     def get_score(self) -> int:
213         return self.score # 現在のスコアを返す
214
215
216 # 制限時間初期化関数
217 ✓ def initialize_timer(time_limit: int) -> tuple:
```

```
118     """
119     タイマーの初期設定
120     引数:
121     time_limit: 制限時間 (秒)
122
123     戻り値:
124     タイマーの開始時刻, 制限時間
125     """
126     start_ticks = pg.time.get_ticks()
127     return start_ticks, time_limit
128
129
130 # 制限時間関数
131 ✓ def show_timer(screen: pg.Surface, font: pg.font.Font, start_ticks: int, time_limit: int) -> bool:
132     """
133     タイマーを表示し、時間切れから3秒後に終了
134     引数:
135     screen: 画面Surface
136     font: 表示用フォント
137     start_ticks: タイマーの開始時刻
138     time_limit: 制限時間
139     戻り値:
140     タイマーが有効かどうか
141     """
142     elapsed_seconds = (pg.time.get_ticks() - start_ticks) / 1000
143     time_left = time_limit - elapsed_seconds
144
145     if time_left > 0:
146         timer_text = font.render(f"Time: {int(time_left)}", True, (255, 255, 255))
147         screen.blit(timer_text, (WIDTH // 2 - timer_text.get_width() // 2, 50))
148         return True # タイマー継続
149     else:
150         # タイマーが終了し、3秒間timeoverを表示して終了
151         if not hasattr(show_timer, "timeover_start"):
152             show_timer.timeover_start = pg.time.get_ticks()
153
154         timeover_text = font.render("timeover", True, (255, 0, 0))
155         screen.blit(timeover_text, (WIDTH // 2 - timeover_text.get_width() // 2, HEIGHT // 2))
156
157         if (pg.time.get_ticks() - show_timer.timeover_start) / 1000 > 3:
158             return False # タイマー終了
159
160     return True
161
162
163 # 盤面領域判定関数
164 ✓ def check_bound(obj_rct: pg.Rect) -> tuple[bool, bool]:
165     """
166     オブジェクトが画面内or画面外を判定し、真理値タプルを返す関数
167     引数: こうかとんやその他動的オブジェクトのRect
168     戻り値: 横方向, 縦方向のはみ出し判定結果 (画面内: True/画面外: False)
169     """
170     yoko, tate = True, True
171     if obj_rct.left < 50 or WIDTH - 50 < obj_rct.right:
172         yoko = False
173     if obj_rct.top < 100 or HEIGHT - 50 < obj_rct.bottom:
174         tate = False
175     for i in range(6):
```



```
276         num = 100 * i
277         if (100 + num) < obj_rct.left < (150 + num) or (100 + num) < obj_rct.right < (150 + num):
278             for j in range(5):
279                 num = 100 * j
280                 if 150 + num < obj_rct.top < 200 + num or 150 + num < obj_rct.bottom < 200 + num:
281                     yoko = False
282                     tate = False
283             return yoko, tate
284
285
286 # ゲームオーバー画面表示関数
287 ✓ def game_over(scr: pg.Surface) -> None:
288     fonto = pg.font.SysFont("hg正楷書体pro", 70)
289     gameover_txt = fonto.render("GAME OVER", True, (255, 0, 0))
290     continue_txt = fonto.render("continue", True, (255, 255, 255))
291     tend_txt = fonto.render("end", True, (255, 255, 255))
292     picture = pg.image.load("images/hoshizora.png") # 画像のパスを修正
293     picture = pg.transform.scale(picture, (WIDTH, HEIGHT)) # 画面サイズにリサイズ
294     scr.blit(picture, (0, 0)) # 背景として画像を描画
295     scr.blit(gameover_txt, [(WIDTH / 2) - (gameover_txt.get_width() / 2), HEIGHT / 4])
296     scr.blit(continue_txt, [(WIDTH / 2) - (continue_txt.get_width() / 2), HEIGHT / 2.3])
297     scr.blit(tend_txt, [(WIDTH / 2) - (tend_txt.get_width() / 2), HEIGHT / 1.8])
298     pg.display.update()
299     time.sleep(5)
300
301
302 # タイトル画面表示関数
303 ✓ def show_title_screen(screen: pg.Surface) -> None:
304     fonto = pg.font.SysFont("hg正楷書体pro", 70)
305     title_txt = fonto.render("ボンバーこうかたん", True, (255, 255, 255))
306     start_txt = fonto.render("START", True, (255, 255, 255))
307     picture = pg.image.load("images/forest_dot1.jpg") # 画像のパスを修正
308
309     while True:
310         screen.blit(picture, (0, 0)) # 背景として画像を描画
311         screen.blit(title_txt, [(WIDTH / 2) - (title_txt.get_width() / 2), HEIGHT / 3])
312         screen.blit(start_txt, [(WIDTH / 2) - (start_txt.get_width() / 2), HEIGHT / 2])
313         pg.display.update()
314
315         for event in pg.event.get():
316             if event.type == pg.QUIT:
317                 pg.quit()
318                 sys.exit()
319             if event.type == pg.KEYDOWN:
320                 return
321
322
323 # 初期位置をランダムに決めるための関数
324 ✓ def random_position() -> list:
325     """
326     盤面領域内の四隅の座標タプルをシャッフルしたリストを返す
327     戻り値: タプルのリスト
328     """
329     pos = [
330         (75, 125),
331         (75, HEIGHT - 75),
332         (WIDTH - 75, 125),
333         (WIDTH - 75, HEIGHT - 75),
334     ]
```

```
334     ]
335     return random.sample(pos, len(pos))
336
337
338 ✓ def main() -> None:
339     """
340     ゲームのメインループを制御する
341     """
342     pg.display.set_caption("ボンバーこうかんとん")
343     screen = pg.display.set_mode((WIDTH, HEIGHT))
344     show_title_screen(screen) # タイトル画面表示
345     bg_img = pg.image.load("images/bg_ver.1.0.png") # 背景(完成版)
346     hero = Hero((75, 125)) # 主人公の初期位置
347     boms = pg.sprite.Group() # 爆弾クラスのグループ作成
348     position = random_position()
349     enemys = pg.sprite.Group() # 敵のスプライトグループ
350     for i, j in enumerate(position[:-1]):
351         enemys.add(Enemy(i, j)) # 敵のインスタンス生成
352     clock = pg.time.Clock()
353     score = Score() # スコアオブジェクトを作成
354     pg.font.init() # フォントの初期化
355     font = pg.font.Font(None, 36) # フォントを作成
356
357     pg.font.init() # フォントの初期化
358     font = pg.font.Font(None, 36) # フォントを作成
359     start_ticks, time_limit = initialize_timer(180)
360
361     while True:
362         for event in pg.event.get():
363             if event.type == pg.QUIT:
364                 return
365             if event.type == pg.KEYDOWN:
366                 if event.key == pg.K_SPACE: # スペースキーで爆弾設置
367                     boms.add(Bomber(hero.rect.center, hero, enemys))
368
369             # 爆弾と敵の衝突判定
370             for bom in boms:
371                 if bom.state == "explosion":
372                     for enemy in enemys:
373                         if bom.rect.colliderect(enemy.rect): # 衝突判定
374                             score.add_score(100) # スコアを加算
375                             enemy.kill() # 敵を消去
376                             break
377
378             # スコアの表示
379             screen.fill((0,0,0), (10,10,150,36))
380             score_text = font.render(f"Score: {score.get_score()}", True, (255, 255, 255))
381             screen.blit(score_text, (10, 10)) # スコアを画面の左上に描画
382             if not show_timer(screen, font, start_ticks, time_limit):
383                 return
384
385             screen.blit(bg_img, [0, 50])
386             key_lst = pg.key.get_pressed()
387             hero.update(key_lst, screen)
388             enemys.update() # 敵グループの更新
389             enemys.draw(screen)
390             boms.update() # 爆弾グループの更新
391             boms.draw(screen)
```

```
392
393         pg.display.update()
394         clock.tick(60) # framerateを60に設定
395
396
397 if __name__ == "__main__":
398     pg.init()
399     main()
400     pg.quit()
401     sys.exit()
```