













 Ryot7m / ProjExD\_05

 |  |  |  |  | 

 Code  Pull requests  Actions  Projects  Wiki  Security  Insights  Settings

 beamplus ▾ ProjExD\_05 / musou\_kokaton.py 

 Go to file t 



Ryot7m ビーム追加機能修正版

7 minutes ago



587 lines (497 loc) · 21.4 KB

Code

Blame

Raw



```
1  import math
2  import random
3  import sys
4  import time
5
6  import pygame as pg
7
8
9  WIDTH = 1600 # ゲームウィンドウの幅
10 HEIGHT = 900 # ゲームウィンドウの高さ
11
12
13 ✓ def check_bound(obj: pg.Rect) -> tuple[bool, bool]:
14     """
15     オブジェクトが画面内か画面外かを判定し、真理値タプルを返す
16     引数 obj: オブジェクト (爆弾, こうかとん, ビーム) SurfaceのRect
17     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)
18     """
19     yoko, tate = True, True
20     if obj.left < 0 or WIDTH < obj.right: # 横方向のはみ出し判定
21         yoko = False
22     if obj.top < 0 or HEIGHT < obj.bottom: # 縦方向のはみ出し判定
23         tate = False
24     return yoko, tate
25
26
27 ✓ def calc_orientation(org: pg.Rect, dst: pg.Rect) -> tuple[float, float]:
28     """
29     orgから見て, dstがどこにあるかを計算し, 方向ベクトルをタプルで返す
30     引数1 org: 爆弾SurfaceのRect
31     引数2 dst: こうかとんSurfaceのRect
32     戻り値: orgから見たdstの方向ベクトルを表すタプル
33     """
34     x_diff, y_diff = dst.centerx-org.centerx, dst.centery-org.centery
35     norm = math.sqrt(x_diff**2+y_diff**2)
36     return x_diff/norm, y_diff/norm
37
38
39 ✓ class Bird(pg.sprite.Sprite):
40     """
41     ゲームキャラクター (こうかとん) に関するクラス
42     """
```

```

43     delta = { # 押下キーと移動量の辞書
44         pg.K_UP: (0, -1),
45         pg.K_DOWN: (0, +1),
46         pg.K_LEFT: (-1, 0),
47         pg.K_RIGHT: (+1, 0),
48     }
49
50 ✓ def __init__(self, num: int, xy: tuple[int, int]):
51     """
52     こうかとおん画像Surfaceを生成する
53     引数1 num: こうかとおん画像ファイル名の番号
54     引数2 xy: こうかとおん画像の位置座標タプル
55     """
56     super().__init__()
57     img0 = pg.transform.rotozoom(pg.image.load(f"ex05/fig/{num}.png"), 0, 2.0)
58     img = pg.transform.flip(img0, True, False) # デフォルトのこうかとおん
59     self.imgs = {
60         (+1, 0): img, # 右
61         (+1, -1): pg.transform.rotozoom(img, 45, 1.0), # 右上
62         (0, -1): pg.transform.rotozoom(img, 90, 1.0), # 上
63         (-1, -1): pg.transform.rotozoom(img0, -45, 1.0), # 左上
64         (-1, 0): img0, # 左
65         (-1, +1): pg.transform.rotozoom(img0, 45, 1.0), # 左下
66         (0, +1): pg.transform.rotozoom(img, -90, 1.0), # 下
67         (+1, +1): pg.transform.rotozoom(img, -45, 1.0), # 右下
68     }
69     self.dire = (+1, 0)
70     self.image = self.imgs[self.dire]
71     self.rect = self.image.get_rect()
72     self.rect.center = xy
73     self.speed = 10
74     self.state = "normal"
75
76
77 ✓ def change_img(self, num: int, screen: pg.Surface):
78     """
79     こうかとおん画像を切り替え、画面に転送する
80     引数1 num: こうかとおん画像ファイル名の番号
81     引数2 screen: 画面Surface
82     """
83     self.image = pg.transform.rotozoom(pg.image.load(f"ex05/fig/{num}.png"), 0, 2.0)
84     screen.blit(self.image, self.rect)
85
86 ✓ def update(self, key_lst: list[bool], screen: pg.Surface):
87     """
88     押下キーに応じてこうかとおんを移動させる
89     引数1 key_lst: 押下キーの真理値リスト
90     引数2 screen: 画面Surface
91     """
92     sum_mv = [0, 0]
93     for k, mv in __class__.delta.items():
94         if key_lst[k]:
95             self.rect.move_ip(+self.speed*mv[0], +self.speed*mv[1])
96             sum_mv[0] += mv[0]
97             sum_mv[1] += mv[1]
98     if check_bound(self.rect) != (True, True):
99         for k, mv in __class__.delta.items():
100             if key_lst[k]:

```

```

101         self.rect.move_ip(-self.speed*mv[0], -self.speed*mv[1])
102     if not (sum_mv[0] == 0 and sum_mv[1] == 0):
103         self.dire = tuple(sum_mv)
104         self.image = self.imgs[self.dire]
105
106
107
108         if self.state == "normal" or self.hyper_life < 0:
109             self.state = "normal"
110             self.image = self.imgs[self.dire]
111         elif self.state == "hyper":
112             self.image = pg.transform.laplacian(self.imgs[self.dire]) # 画像imageを変換
113             self.hyper_life -= 1 # 発動時間hyper_lifeを1減らす
114
115     screen.blit(self.image, self.rect)
116
117     def get_direction(self) -> tuple[int, int]:
118         return self.dire
119
120     def change_state(self, state: str, hyper_life: int):
121         """
122         追加機能3
123         引数1 state : 状態 ("hyper"と"normal")
124         引数2 hyper_life : 発動時間
125         """
126         self.state = state
127         self.hyper_life = hyper_life
128
129     class Bomb(pg.sprite.Sprite):
130         """
131         爆弾に関するクラス
132         """
133         colors = [(255, 0, 0), (0, 255, 0), (0, 0, 255), (255, 255, 0), (255, 0, 255), (0, 255, 255)]
134
135     def __init__(self, emy: "Enemy", bird: Bird):
136         """
137         爆弾円Surfaceを生成する
138         引数1 emy : 爆弾を投下する敵機
139         引数2 bird : 攻撃対象のこうかとん
140         """
141         super().__init__()
142         rad = random.randint(10, 50) # 爆弾円の半径 : 10以上50以下の乱数
143         color = random.choice(__class__.colors) # 爆弾円の色 : クラス変数からランダム選択
144         self.image = pg.Surface((2*rad, 2*rad))
145         pg.draw.circle(self.image, color, (rad, rad), rad)
146         self.image.set_colorkey((0, 0, 0))
147         self.rect = self.image.get_rect()
148         # 爆弾を投下するemyから見た攻撃対象のbirdの方向を計算
149         self.vx, self.vy = calc_orientation(emy.rect, bird.rect)
150         self.rect.centerx = emy.rect.centerx
151         self.rect.centery = emy.rect.centery + emy.rect.height/2
152         self.speed = 6
153
154     def update(self):
155         """
156         爆弾を速度ベクトルself.vx, self.vyに基づき移動させる
157         引数 screen : 画面Surface
158         """

```

```
159         self.rect.move_ip(+self.speed*self.vx, +self.speed*self.vy)
160         if check_bound(self.rect) != (True, True):
161             self.kill()
162
163
164 ✓ class Beam(pg.sprite.Sprite):
165     """
166     ビームに関するクラス
167     """
168 ✓ def __init__(self, bird: Bird, angle_a: float=0):
169     """
170     ビーム画像Surfaceを生成する
171     引数 bird: ビームを放つこうかとん
172     """
173     super().__init__()
174     self.vx, self.vy = bird.get_direction()
175     angle = math.degrees(math.atan2(-self.vy, self.vx))+angle_a
176     self.size = random.uniform(1.5, 3.0)
177     self.image = pg.transform.rotozoom(pg.image.load(f"ex05/fig/beam.png"), angle, self.size)
178     self.vx = math.cos(math.radians(angle))
179     self.vy = -math.sin(math.radians(angle))
180     self.rect = self.image.get_rect()
181     self.rect.centery = bird.rect.centery+bird.rect.height*self.vy
182     self.rect.centerx = bird.rect.centerx+bird.rect.width*self.vx
183     self.speed = random.uniform(5, 20) #ビームのスピードをランダムに変更
184
185 ✓ def update(self):
186     """
187     ビームを速度ベクトルself.vx, self.vyに基づき移動させる
188     引数 screen: 画面Surface
189     """
190     self.rect.move_ip(+self.speed*self.vx, +self.speed*self.vy)
191     if check_bound(self.rect) != (True, True):
192         self.kill()
193
194
195 ✓ class Explosion(pg.sprite.Sprite):
196     """
197     爆発に関するクラス
198     """
199 ✓ def __init__(self, obj: "Bomb|Enemy", life: int):
200     """
201     爆弾が爆発するエフェクトを生成する
202     引数1 obj: 爆発するBombまたは敵機インスタンス
203     引数2 life: 爆発時間
204     """
205     super().__init__()
206     img = pg.image.load("ex05/fig/explosion.gif")
207     self.imgs = [img, pg.transform.flip(img, 1, 1)]
208     self.image = self.imgs[0]
209     self.rect = self.image.get_rect(center=obj.rect.center)
210     self.life = life
211
212 ✓ def update(self):
213     """
214     爆発時間を1減算した爆発経過時間_lifeに応じて爆発画像を切り替えることで
215     爆発エフェクトを表現する
216     """
```

```

217         self.life -= 1
218         self.image = self.imgs[self.life//10%2]
219         if self.life < 0:
220             self.kill()
221
222
223 ✓ class Enemy(pg.sprite.Sprite):
224     """
225     敵機に関するクラス
226     """
227     imgs = [pg.image.load(f"ex05/fig/alien{i}.png") for i in range(1, 4)]
228
229 ✓ def __init__(self):
230     super().__init__()
231     self.image = random.choice(__class__.imgs)
232     self.rect = self.image.get_rect()
233     self.rect.center = random.randint(0, WIDTH), 0
234     self.vy = +6
235     self.bound = random.randint(50, HEIGHT/2) # 停止位置
236     self.state = "down" # 降下状態or停止状態
237     self.interval = random.randint(50, 300) # 爆弾投下インターバル
238
239 ✓ def update(self):
240     """
241     敵機を速度ベクトルself.vyに基づき移動（降下）させる
242     ランダムに決めた停止位置_boundまで降下したら、_stateを停止状態に変更する
243     引数 screen: 画面Surface
244     """
245     if self.rect.centery > self.bound:
246         self.vy = 0
247         self.state = "stop"
248         self.rect.centery += self.vy
249
250
251 ✓ class Shield(pg.sprite.Sprite):
252     """
253     防御壁に関するクラス
254     引数1 bird: 防御壁
255     引数2 life: 防御壁の発動時間
256     """
257 ✓ def __init__(self, bird: Bird, life: int):
258     super().__init__()
259     self.vx, self.vy = bird.get_direction()
260     theta = math.atan2(-self.vy, self.vx) # こうかとの向き（弧度法）
261     angle = math.degrees(theta) # こうかとの向き（度数法）
262     self.image = pg.Surface((20, bird.rect.height*2))
263     self.image = pg.transform.rotozoom(self.image, angle, 1.0)
264     pg.draw.rect(self.image, (0, 0, 0), pg.Rect(0, 0, 20, bird.rect.height*2))
265     self.rect = self.image.get_rect()
266     self.rect.centerx = bird.rect.centerx+bird.rect.width*self.vx
267     self.rect.centery = bird.rect.centery+bird.rect.height*self.vy
268     self.life = life
269     # 爆弾を投下するemyから見た攻撃対象のbirdの方向を計算
270
271 ✓ def update(self):
272     """
273     爆発時間を1減算し、発動時間中は防御壁矩形を有効化
274     """

```

```
275         self.life -= 1
276         if self.life < 0:
277             self.kill()
278
279
280 ✓ class Score:
281     """
282     打ち落とした爆弾、敵機の数スコアとして表示するクラス
283     爆弾: 1点
284     敵機: 10点
285     """
286 ✓ def __init__(self):
287     self.font = pg.font.Font(None, 50)
288     self.color = (0, 0, 255)
289     self.score = 0
290     self.image = self.font.render(f"Score: {self.score}", 0, self.color)
291     self.rect = self.image.get_rect()
292     self.rect.center = 100, HEIGHT-50
293
294     def score_up(self, add):
295         self.score += add
296
297     def update(self, screen: pg.Surface):
298         self.image = self.font.render(f"Score: {self.score}", 0, self.color)
299         screen.blit(self.image, self.rect)
300
301
302 ✓ class NeoGravity(pg.sprite.Sprite):
303 ✓ def __init__(self, life: int):
304     super().__init__()
305     self.image = pg.Surface((WIDTH, HEIGHT))
306     pg.draw.rect(self.image, (10, 10, 10), pg.Rect(0, 0, WIDTH, HEIGHT))
307     self.image.set_colorkey((0, 0, 0))
308     self.image.set_alpha(200)
309     self.rect = self.image.get_rect()
310     self.rect.center = WIDTH/2, HEIGHT/2
311     self.life = life
312
313     def update(self):
314         self.life -= 1
315         if self.life < 0:
316             self.kill()
317
318 ✓ class Gravity(pg.sprite.Sprite):
319 ✓ def __init__(self, bird, life):
320     super().__init__()
321     rad = 200
322     self.life = life
323     self.image = pg.Surface((2*rad, 2*rad))
324     pg.draw.circle(self.image, (1, 1, 1), (rad, rad), rad)
325     self.rect = self.image.get_rect()
326     self.image.set_colorkey("black")
327     self.image.set_alpha(127) #黒を透明化
328     self.rect.center = bird.rect.center #self.rectがこうかとんを追う
329
330
331
332 ✓ def update(self, bird):
333     self.rect.center = bird.rect.center
```

```

333         self.rect.center = bird.rect.center
334         self.life = self.life - 1
335         if self.life <= 0:
336             self.kill()
337
338 ✓ class BeamPlus(pg.sprite.Sprite):
339     """
340     2発のビームの発射を可能にするクラス
341     """
342 ✓ def __init__(self, bird: Bird):
343     """
344     ビーム画像Surfaceを生成する
345     引数 bird: ビームを放つこうかとん
346     """
347     super().__init__()
348     self.vx, self.vy = bird.get_direction()
349     angle = math.degrees(math.atan2(-self.vy, self.vx))
350     self.image = pg.Surface((bird.rect.height/2, 20))
351     self.size = random.uniform(0.1, 1.0) #ビームの区別をつけるため小さくしている
352     self.image = pg.transform.rotozoom(self.image, angle, self.size)
353     pg.draw.rect(self.image, (random.randint(0, 255), random.randint(0, 255), random.randint(0, 255))
354     self.vx = math.cos(math.radians(angle))
355     self.vy = -math.sin(math.radians(angle))
356     self.rect = self.image.get_rect()
357     self.rect.centery = bird.rect.centery+bird.rect.height*self.vy
358     self.rect.centerx = bird.rect.centerx+bird.rect.width*self.vx
359     self.speed = 30 #大きさを小さくした分性能の差を無くするためにスピードを上げる
360
361 ✓ def update(self):
362     """
363     ビームを速度ベクトルself.vx, self.vyに基づき移動させる
364     引数 screen: 画面Surface
365     """
366     self.rect.move_ip(+self.speed*self.vx, +self.speed*self.vy)
367     if check_bound(self.rect) != (True, True):
368         self.kill()
369
370
371 ✓ class Beamplusalpha:
372     """
373     全方向に速度が不特定のビームを放つ処理
374     """
375 ✓ def __init__(self, bird: Bird, num: int):
376     """
377     ビーム画像Surfaceを生成する
378     引数 bird: ビームを放つこうかとん
379     引数 num: 発射するビームの数
380     bird, numの初期化を行う
381     """
382     self.beam_list = [] #リストの生成
383     self.bird = bird
384     self.num = num
385 ✓ def gen_beams(self):
386     """
387     角度をつけてビームを出す処理
388     """
389     vx, vy = self.bird.get_direction()
390     for i in range(-180, 181, int(100/(self.num-1))): #-180度から180度の間でint(100/(self.num-1))おきに
391         self.beam_list.append(BeamPlus(self.bird, i)) #ビームの値をリストに代入

```

```

392         return self.beam_list
393
394
395 class Levelup:
396     def __init__(self):
397         """
398         ビームの結果に応じてレベルの上がる処理
399         """
400         self.font = pg.font.Font(None, 50)
401         self.color = (247, 146, 19)
402         self.level = 0
403         self.image = self.font.render(f"LEVEL: {self.level}", 0, self.color) #現在のレベル表示
404         self.rect = self.image.get_rect()
405         self.rect.center = WIDTH-80, 30 #self.imageの内容の表示位置
406
407     def levelup(self, add):
408         """
409         スコア増加の処理
410         """
411         self.level += add
412
413     def update(self, screen: pg.Surface):
414         self.image = self.font.render(f"LEVEL: {self.level}", 0, self.color)
415         screen.blit(self.image, self.rect)
416
417 def main():
418     pg.display.set_caption("真！ どうかとん無双")
419     screen = pg.display.set_mode((WIDTH, HEIGHT))
420     bg_img = pg.image.load("ex05/fig/pg_bg.jpg")
421     score = Score()
422
423     bird = Bird(3, (900, 400))
424     bombs = pg.sprite.Group()
425     beams = pg.sprite.Group()
426     exps = pg.sprite.Group()
427     emys = pg.sprite.Group()
428     neogrs = pg.sprite.Group()
429     gravities = pg.sprite.Group()
430     pluses = pg.sprite.Group()
431     levels = Levelup()
432     levels.level = 1
433
434     score.score = 0
435
436     shields = pg.sprite.Group()
437
438
439     tmr = 0
440     clock = pg.time.Clock()
441     while True:
442         key_lst = pg.key.get_pressed()
443         for event in pg.event.get():
444             if event.type == pg.QUIT:
445                 return 0
446             if event.type == pg.KEYDOWN and event.key == pg.K_SPACE:
447                 beams.add(Beam(bird))
448                 for i in range(1,100):
449                     if score.score >= i*10:

```



```
450         pluses.add(BeamPlus(bird))
451
452
453     if event.type == pg.KEYDOWN and event.key == pg.K_LSHIFT: # 左シフトが押されているか判定
454         bird.speed = 20 # スピードアップ
455     if event.type == pg.KEYUP and event.key == pg.K_LSHIFT: # 左シフトが押された状態から離れたら
456         bird.speed = 10 # もとのスピードに戻る
457     if event.type == pg.KEYDOWN and event.key == pg.K_RETURN:
458         if score.score > 200:
459             neogrs.add(NeoGravity(400))
460             score.score_up(-200)
461 # 追加機能3
462     if event.type == pg.KEYDOWN and event.key == pg.K_RSHIFT and score.score > 100: #→Shiftキー
463         score.score -= 100
464         bird.change_state("hyper", 500)
465
466     if event.type == pg.KEYDOWN and event.key == pg.K_TAB and score.score > 50:
467         # 矢印キーとtabキーが押されて、スコアが50以上ならスコアを-50する。
468         gravities.add(Gravity(bird, 500))
469         score.score -= 50
470
471     if event.type == pg.KEYDOWN and event.key == pg.K_CAPSLOCK and len(shields) == 0 :
472         if score.score > 50:
473             score.score_up(-50)
474             shields.add(Shield(bird, 400))
475
476     if event.type == pg.KEYDOWN and event.key == pg.K_F1 and score.score > 40:
477         levels.levelup(3) # レベル3アップ
478         beams.add(Beamplusalpha(bird, 6).gen_beams())
479         score.score_up(-40)
480
481
482     screen.blit(bg_img, [0, 0])
483
484
485     if tmr%200 == 0: # 200フレームに1回、敵機を出現させる
486         emys.add(Enemy())
487
488     for emy in emys:
489         if emy.state == "stop" and tmr%emy.interval == 0:
490             # 敵機が停止状態に入ったら、intervalに応じて爆弾投下
491             bombs.add(Bomb(emy, bird))
492
493     for emy in pg.sprite.groupcollide(emys, beams, True, True).keys():
494         exps.add(Explosion(emy, 100)) # 爆発エフェクト
495         score.score_up(10) # 10点アップ
496         levels.levelup(1) # レベル1アップ
497         bird.change_img(6, screen) # こうかたん喜びエフェクト
498
499     for bomb in pg.sprite.groupcollide(bombs, beams, True, True).keys():
500         exps.add(Explosion(bomb, 50)) # 爆発エフェクト
501         score.score_up(1) # 1点アップ
502
503
504     for bomb in pg.sprite.spritecollide(bird, bombs, True):
505         if bird.state == "normal":
506             bird.change_img(8, screen) # こうかたん悲しみエフェクト
507         score.update(screen)
```

```
508         pg.display.update()
509         time.sleep(2)
510         return
511     elif bird.state == "hyper":
512         exps.add(Explosion(bomb, 50)) # 爆発エフェクト
513         score.score_up(1) # 1点アップ
514
515
516     for emy in pg.sprite.groupcollide(emy, gravities, True, False).keys():
517         exps.add(Explosion(emy, 100)) # 爆発エフェクト
518         score.score_up(10) # 10点アップ
519         levels.levelup(1) # レベルが1上がる
520         bird.change_img(6, screen) # こうかたん喜びエフェクト
521
522     # for bomb in pg.sprite.groupcollide(bombs, gravities, True, False).keys():
523
524     for bomb in pg.sprite.groupcollide(bombs, shields, True, False).keys():
525
526         exps.add(Explosion(bomb, 50)) # 爆発エフェクト
527         score.score_up(1) # 1点アップ
528
529     if len(pg.sprite.spritecollide(bird, bombs, True)) != 0:
530         bird.change_img(8, screen) # こうかたん悲しみエフェクト
531         score.update(screen)
532         pg.display.update()
533         time.sleep(2)
534         return
535
536     for bomb in pg.sprite.groupcollide(bombs, neogrs, True, False).keys():
537         exps.add(Explosion(bomb, 50)) # 爆発エフェクト
538         score.score_up(1)
539
540     for emy in pg.sprite.groupcollide(emy, neogrs, True, False).keys():
541         exps.add(Explosion(emy, 100)) # 爆発エフェクト
542         score.score_up(10) # 10点アップ
543         bird.change_img(6, screen) # こうかたん喜びエフェクト
544
545     for emy in pg.sprite.groupcollide(emy, pluses, True, True).keys():
546         exps.add(Explosion(emy, 100)) # 爆発エフェクト
547         levels.levelup(1) # レベルが1上がる
548         score.score_up(10) # 10点アップ
549         bird.change_img(6, screen) # こうかたん喜びエフェクト
550
551     for bomb in pg.sprite.groupcollide(bombs, pluses, True, True).keys():
552         exps.add(Explosion(bomb, 50)) # 爆発エフェクト
553         score.score_up(1) # 1点アップ
554
555     bird.update(key_lst, screen)
556     beams.update()
557     beams.draw(screen)
558     emys.update()
559     emys.draw(screen)
560     bombs.update()
561     bombs.draw(screen)
562     exps.update()
563     exps.draw(screen)
564     neogrs.update()
565     neogrs.draw(screen)
```

```
566         score.update(screen)
567         gravities.update(bird)
568         gravities.draw(screen)
569
570         shields.update()
571         shields.draw(screen)
572         pluses.update()
573         pluses.draw(screen)
574         levels.update(screen)
575
576         pg.display.update()
577         tmr += 1
578         clock.tick(50)
579
580
581 if __name__ == "__main__":
582     pg.init()
583     main()
584     pg.quit()
585     sys.exit
586     sys.exit()
```