

forked from [omochio/ProjExD_05](#)

main

ProjExD_05-1 / README.md

Go to file

t

...

10 minutes ago

Raw    ▼ 

実行環境の必要条件

- ## ゲームの概要

操作方法

- ## スコア

- 横に進む
- 敵にタックル

- 時間経過

クリア条件

クリアは存在せず, どこまで進めるか自分の限界に挑戦する
穴に落ちるとゲームオーバー

ゲームの実装

共通基本機能

- プレイヤーに関するクラス
- ブロックに関するクラス

担当追加機能

C0A22094

- プレイヤーの加速度運動
- プレイヤーとブロックの衝突
- レベルの生成と保持
- スクロール
- ゲームオーバー
- 敵の自動生成

C0B21082

- ゲーム内のスコア加算、表示

C0B22092


- エネミークラスの実装

C0B22096

- プレイヤーの無敵状態


C0B22108


- プレイヤーが投げるBox
 - Boxの当たり判定
 - 物理特性再現
- プレイヤーと敵が投げるBombとExplode
- 敵の自動攻撃
- 距離をSCOREに反映


 **c0b21082 / ProjExD_05-1** Public


forked from [omochio/ProjExD_05](#)


<> Code


 Pull requests


 Actions

 Projects

 Wiki

 Security

 Insights

 Settings

 **main**

ProjExD_05-1 / main.py 

Go to file

t

...

 **omochio** ゲームオーバーの呼び出しを作成


19 minutes ago  


847 lines (728 loc) · 30.4 KB


Code


Blame

Raw









```
1 import sys
2 import random
3 import pygame as pg
4
5 WIDTH = 1600
6 HEIGHT = 1000
7 # ビューの座標
8 VIEW_POS = (WIDTH // 2, HEIGHT - 200)
9
10 # スクロールのために動的に変更されるrectのリスト
11 dynamic_rect_lst = []
12
13 class Player(pg.sprite.Sprite):
14     """
15     Playerに関するクラス
16     """
17     # 入力と移動方向の対応
18     __move_dict = {
19         pg.K_LEFT: (-1, 0),
20         pg.K_a: (-1, 0),
21         pg.K_RIGHT: (1, 0),
22         pg.K_d: (1, 0),
23         pg.K_UP: (0, -1),
24         pg.K_SPACE: (0, -1)
25     }
26
27     def __init__(self, center: tuple[int, int]):
28         """
29         Playerクラスの初期化
30         center: 初期座標
31         """
32         super().__init__()
33         self.__size = (64, 64)
34         self.image = pg.Surface(self.__size)
35         self.image.fill((255, 255, 255))
36         self.rect = self.image.get_rect()
37         self.rect.center = center
38         self.my_timer = 0
39         self.box_timer = 0
40         self.curve_timer = 0
```

Symbols

×

Find definitions and references for functions and other symbols in this file by clicking a symbol below or in the code.

Filter symbols

r

const

WIDTH

const

HEIGHT

const

VIEW_POS

const

dynamic_rect_lst

class

Player

func

__init__

func

is_grounded

func

is_grounded

func

vel

func

set_vel

func

add_vel

func

change_state

func

check_hyper

func

update

func

update_box

func

update_bomb

func

update_throw_predict

class

Block

func

__init__

func

size

class

Box

func

__init__

```
42         self.is_pre_predict = False
43         self.__acc = [.0, .0]
44         self.__vel = [.0, .0]
45         self.__gravity_acc = 1
46         self.__walk_acc = 2
47         self.__walk_vel_max = 10
48         self.__jump_init_vel = 20
49         self.__is_grounded = False
50         self.state = "normal" # プレイヤーの状態
51         self.hyper_life = 0 # 残りの無敵状態時間
52
53     @property
54     def is_grounded(self) -> bool:
55         """
56         接地判定変数のgetter
57         返回值: 接地判定変数の値
58         """
59         return self.__is_grounded
60
61     @is_grounded.setter
62     def is_grounded(self, value: bool):
63         """
64         接地判定変数のsetter
65         value: 接地判定変数の値
66         """
67         self.__is_grounded = value
68
69     @property
70     def vel(self) -> list[float, float]:
71         """
72         速度のgetter
73         返回值: 速度のリスト
74         """
75         return self.__vel.copy()
76
77     def set_vel(self, vx: float = None, vy: float = None):
78         """
79         速度のsetter
80         Noneを入れた方向は変更しない
81         vx: x方向の速度
82         vy: y方向の速度
83         """
84         if vx is not None:
85             self.__vel[0] = vx
86         if vy is not None:
87             self.__vel[1] = vy
88
89     def add_vel(self, vx: float = .0, vy: float = .0):
90         """
91         速度の加算
92         vx: x方向の加算速度
93         vy: y方向の加算速度
94         """
95         self.__vel[0] += vx
96         self.__vel[1] += vy
97
98     def change_state(self, state: str, hyper_life: int):
99         """
```

```
func update
func set_vel

func is_moving

class Bomb
    func __init__
    func update
    func set_vel
```

```

100         右シフトキーが押された時に、プレイヤーを無敵状態に
101         引数1 state : プレイヤーの状態
102         引数2 hyper_life : 無敵状態になっている時間
103         戻り値 : なし
104         """
105         self.state = state
106         self.hyper_life = hyper_life
107
108     def check_hyper(self):
109         """
110         プレイヤーが無敵状態かどうかを判定し、プレイヤーの
111         戻り値 : なし
112         """
113         if self.state == "hyper":
114             # プレイヤーが無敵状態だったら
115             self.image.fill((168, 88, 168)) # プレイヤ
116             self.hyper_life += -1 # 残りの無敵状態時間
117
118         if self.hyper_life < 0: # 残りの無敵状態時間が0
119             self.state == "normal" # プレイヤーを通常状
120             self.image.fill((255, 255, 255)) # プレイ
121
122
123     def update(self, key_lst: dict):
124         """
125         Playerの更新を行う
126         key_lst: 押されているキーのリスト
127         """
128
129         self.my_timer += 1
130         self.update_box(key_lst)
131         self.update_bomb(key_lst)
132         self.update_throw_predict(key_lst)
133         self.__acc = [.0, .0]
134         # 入力と移動方向dictに応じて加速度を設定
135         for d in __class__.__move_dict:
136             if key_lst[d]:
137                 self.__acc[0] += self.__walk_acc * __
138                 # 接地時のみジャンプ可能
139                 if self.is_grounded:
140                     self.set_vel(vy=self.__jump_init_
141                     if self.vel[1] < 0:
142                         self.is_grounded = False
143
144         # 重力加速度を加算
145         if not self.is_grounded:
146             self.__acc[1] += self.__gravity_acc
147
148         # 加速度と速度上限から速度を計算
149         self.add_vel(self.__acc[0])
150         if self.vel[0] < -self.__walk_vel_max:
151             self.set_vel(-self.__walk_vel_max)
152         elif self.vel[0] > self.__walk_vel_max:
153             self.set_vel(self.__walk_vel_max)
154         self.add_vel(vy=self.__acc[1])
155
156         self.check_hyper()
157

```

```
158
159 ✓ def update_box(self, key_lst: dict):
160     """
161     Press mouse Left
162     box throw
163     """
164
165     #次に投げれるようになるまでのフレーム数
166     if self.my_timer - self.box_timer < 10:
167         return
168
169
170     pg.event.get()
171     if pg.mouse.get_pressed()[0]:
172         self.box_timer = self.my_timer
173         throw_arg = [0, 0]
174         mouse_pos = list(pg.mouse.get_pos())
175         player_pos = list(self.rect.center)
176         throw_arg[0] = (mouse_pos[0] - player_pos[0])
177         throw_arg[1] = (mouse_pos[1] - player_pos[1])
178         Box((self.rect.centerx + throw_arg[0], self.rect.centery + throw_arg[1]), self.box_size, self.box_color)
179
180
181
182 ✓ def update_bomb(self, key_lst: dict):
183     """
184     Press mouse Right
185     bomb throw
186     """
187
188     #次に投げれるようになるまでのフレーム数
189     if self.my_timer - self.bomb_timer < 30:
190         return
191
192
193     pg.event.get()
194     if pg.mouse.get_pressed()[2]:
195         self.bomb_timer = self.my_timer
196         throw_arg = [0, 0]
197         mouse_pos = list(pg.mouse.get_pos())
198         player_pos = list(self.rect.center)
199         throw_arg[0] = (mouse_pos[0] - player_pos[0])
200         throw_arg[1] = (mouse_pos[1] - player_pos[1])
201         Bomb(self.rect.center, tuple(throw_arg), self.bomb_size, self.bomb_color)
202
203 ✓ def update_throw_predict(self, key_lst: dict):
204     """
205     Press Shift
206     draw throw curve
207     """
208
209
210     pg.event.get()
211     #CTRLで予測線
212     if (key_lst[pg.K_RCTRL]):
213         if not self.is_pre_predict:
214             self.is_predict = not self.is_predict
215             self.is_pre_predict = True
```

```
216         else:
217             self.is_pre_predict = False
218
219         #次に投げれるようになるまでのフレーム数
220         if self.my_timer - self.curve_timer < 10:
221             return
222
223         if self.is_predict:
224             self.curve_timer = self.my_timer
225             throw_arg = [0,0]
226             mouse_pos = list(pg.mouse.get_pos())
227             player_pos = list(self.rect.center)
228             throw_arg[0] = (mouse_pos[0] - player_pos[0])
229             throw_arg[1] = (mouse_pos[1] - player_pos[1])
230             Throw_predict(self.rect.center,tuple(throw_arg))
231
232 ✓ class Block(pg.sprite.Sprite):
233     """
234     初期生成されるブロックに関するクラス
235     """
236 ✓ def __init__(self, center: tuple[int, int], size: tuple[int, int]):
237     super().__init__()
238     self.__size = size
239     self.image = pg.Surface(size)
240     self.image.fill((127, 127, 127))
241     self.rect = self.image.get_rect()
242     self.rect.center = center
243
244     @property
245 ✓ def size(self) -> tuple[int, int]:
246     """
247     サイズのgetter
248     返回值: サイズのタプル
249     """
250     return self.__size
251
252 ✓ class Box(pg.sprite.Sprite):
253     """
254     playerがなげるBoxClassです
255     """
256     boxes = pg.sprite.Group()
257 ✓ def __init__(self, pos: tuple[int, int],vel:tuple[int, int]):
258     global dynamic_rect_lst
259     super().__init__()
260     self.image = pg.Surface((50, 50))
261     self.image.fill((0, 255, 255))
262     self.rect = self.image.get_rect()
263     self.rect.center = pos
264     self.gravity_val = 1
265     self.life = 0
266     self.is_ground = False
267     self.vel = list(vel)
268     self.acc = [0,0]
269     self.acc = [0,self.gravity_val]
270     __class__.boxes.add(self)
271     dynamic_rect_lst.append(self.rect)
272
273
```

```

274 ✓ def update(self):
275
276     self.life += 1
277     if self.life > 6000:
278         self.kill()
279     self.vel[0] += self.acc[0]
280     self.vel[1] += self.acc[1]
281
282
283
284     if self.is_ground:
285         self.vel[1] = 0
286         self.vel[0] = 0
287
288     self.rect.x += self.vel[0]
289     self.rect.y += self.vel[1]
290
291 def set_vel(self, vx, vy):
292     self.vel[1] = vy
293     self.vel[0] = vx
294
295 def is_moving(self):
296     #[0,0]でないならFalse
297     return not self.vel == [0,0]
298
299 ✓ class Bomb(pg.sprite.Sprite):
300     """
301     playerがなげるBombClassです
302     """
303     bombs = pg.sprite.Group()
304 ✓ def __init__(self, pos: tuple[int, int], vel: tuple
305     global dynamic_rect_lst
306     super().__init__()
307     self.image = pg.Surface((30, 30))
308     self.image.fill((255, 128, 0))
309     self.rect = self.image.get_rect()
310     #self.image.set_alpha(128)
311     self.rect.center = pos
312     self.gravity_val = 1
313     self.life = 0
314     self.is_ground = False
315     self.vel = list(vel)
316     self.acc = [0,0]
317     self.acc = [0,self.gravity_val]
318     __class__.bombs.add(self)
319     dynamic_rect_lst.append(self.rect)
320
321 ✓ def update(self):
322     life_max = 180
323     self.life += 1
324
325     #自動で消えるまでの時間
326     if self.life >= life_max:
327         Explode(self.rect.center)
328         self.kill()
329
330     #爆発までの時間を色で表現
331     self.image.fill((255 - 128*int((self.life/lif

```



```

332
333         self.vel[0] += self.acc[0]
334         self.vel[1] += self.acc[1]
335
336
337
338         if self.is_ground:
339             self.vel[1] = 0
340             self.vel[0] = 0
341
342         self.rect.x += self.vel[0]
343         self.rect.y += self.vel[1]
344
345     def set_vel(self, vx, vy):
346         self.vel[1] = vy
347         self.vel[0] = vx
348
349 ✓ class Explode(pg.sprite.Sprite):
350     """
351     Bombが爆発した時に呼び出されるExplodeClassです
352     """
353     explodes = pg.sprite.Group()
354 ✓ def __init__(self, pos: tuple[int, int], power: flo
355         global dynamic_rect_lst
356         super().__init__()
357         rad = power * 16
358         self.image = pg.Surface((rad, rad))
359         self.image.fill((200, 0, 0))
360         pg.draw.circle(self.image, (200, 0, 0), (rad,
361         self.image.set_colorkey((255, 255, 255))
362         self.image.set_alpha(128)
363         self.rect = self.image.get_rect()
364         self.rect.center = pos
365         self.life = 0
366         __class__.explodes.add(self)
367         dynamic_rect_lst.append(self.rect)
368
369 ✓ def update(self):
370     self.life += 1
371     #自動で消えるまでの時間
372     if self.life > 12:
373         self.kill()
374
375 ✓ class Throw_predict(pg.sprite.Sprite):
376     """
377     playerがなげるものの予測線Classです
378     """
379     predicts = pg.sprite.Group()
380 ✓ def __init__(self, pos: tuple[int, int], vel: tuple
381         global dynamic_rect_lst
382         super().__init__()
383         self.image = pg.Surface((15, 15))
384         self.image.fill((255, 200, 255))
385         self.rect = self.image.get_rect()
386         self.rect.center = pos
387         self.gravity_val = 1
388         self.life = 0
389         self.vel = list(vel)
390         self.predicts.add(self)

```

```

390         self.acc = [0,0]
391         self.acc = [0,self.gravity_val]
392         __class__.predicts.add(self)
393         dynamic_rect_lst.append(self.rect)
394
395     def update(self):
396
397         self.life += 1
398         #自動で消えるまでの時間
399         if self.life > 20:
400             self.kill()
401             self.vel[0] += self.acc[0]
402             self.vel[1] += self.acc[1]
403
404             self.rect.x += self.vel[0]
405             self.rect.y += self.vel[1]
406
407     def set_vel(self,vx,vy):
408         self.vel[1] = vy
409         self.vel[0] = vx
410
411     class Enemy(pg.sprite.Sprite): # エネミークラス
412         x = 400
413         y = 700
414     def __init__(self, center: tuple[int, int]):
415         global dynamic_rect_lst
416         super().__init__()
417         self.image = pg.Surface((64, 64))
418         self.image.fill((255, 0, 0))
419         self.rect = self.image.get_rect()
420         dynamic_rect_lst.append(self.rect)
421         self.rect.center = center
422         self.life = 0
423
424     def update(self):
425         global VIEW_POS
426         if self.life % 60 == 0 and 0 <= self.rect.cer
427             self.throw_bomb()
428             self.life += 1
429
430     def throw_bomb(self):
431         throw_arg = [0,0]
432         player_pos = list(VIEW_POS)
433         enemy_pos = list(self.rect.center)
434         throw_arg[0] = (player_pos[0] - enemy_pos[0])
435         throw_arg[1] = (player_pos[1] - enemy_pos[1])
436         Bomb(self.rect.center,tuple(throw_arg),power=
437
438     class Level():
439         """
440         レベル生成と保持を担うクラス
441         """
442     def __init__(self):
443         global dynamic_rect_lst
444         self.blocks = pg.sprite.Group()
445         self.__fllcl_height = 100 # 床と天井の高さ
446         self.__ceil_y = -HEIGHT // 2 # 天井の中心y座
447         # 床
448         self.min_floor_width = 100

```

```

448         self.min_floor_width = 100
449         self.max_floor_width = WIDTH // 2
450         # 天井の生成
451         self.create_ceil((WIDTH // 2, self.__ceil_y))
452         # 床の生成
453         self.blocks.add(Block((WIDTH // 2, HEIGHT), (
454         dynamic_rect_lst.append(self.blocks.sprites())
455         self.__left_floor_rct = self.blocks.sprites()
456         self.__right_floor_rct = self.blocks.sprites()
457
458         # 障害物
459         self.min_obstacle_count = 50
460         self.max_obstacle_count = 100
461         self.min_obstacle_width = 50
462         self.min_obstacle_height = 50
463         self.max_obstacle_width = 100
464         self.max_obstacle_height = 100
465
466         # 穴
467         self.min_hole_width = 0
468         self.max_hole_width = WIDTH // 2
469
470         # 敵
471         self.enemies = pg.sprite.Group()
472         self.min_enemy_count = 10
473         self.max_enemy_count = 20
474
475     def update(self):
476         """
477         レベルの更新を行う
478         """
479         global WIDTH
480         # 左端の床のx座標が-WIDTH//2より大きくなったら生
481         if self.__left_floor_rct.left >= -WIDTH // 2:
482             self.create_ceil((self.__left_floor_rct.l
483             prev_floor_rct = self.__left_floor_rct
484             total = 0
485             # 生成した床の長さが穴を含めてWIDTHを超えるま
486             while total < WIDTH:
487                 offset = random.randint(self.min_hole
488                 sizex = random.randint(self.min_floor
489                 if total + offset + sizex >= WIDTH:
490                     sizex = WIDTH - total
491                     offset = 0
492                     total += sizex
493                 else:
494                     total += offset + sizex
495                 self.create_floor((self.__left_floor_
496                 self.__left_floor_rct = self.blocks.s
497                 self.create_obstacles((self.__left_floor_
498                 self.create_enemies((self.__left_floor_rc
499         # 右端の床のx座標がWIDTH * 3//2より小さくなったら
500         elif self.__right_floor_rct.right <= WIDTH *
501             self.create_ceil((self.__right_floor_rct.
502             prev_floor_rct = self.__right_floor_rct
503             total = 0
504             # 生成した床の長さが穴を含めてWIDTHを超えるま
505             while total < WIDTH:
506                 offset = random.randint(self.min_hole

```

```

506         offset = random.randint(self.min_offset, self.max_offset)
507         sizex = random.randint(self.min_floor, self.max_floor)
508         if total + offset + sizex >= WIDTH:
509             sizex = WIDTH - total
510             offset = 0
511             total += sizex
512         else:
513             total += offset + sizex
514         self.create_floor((self.__right_floor_center, self.__right_floor_center_y))
515         self.__right_floor_rct = self.blocks.sprites()[0].rect
516         self.create_obstacles((prev_floor_rct.right, prev_floor_center_y,
517                               prev_floor_rct.right, prev_floor_center_y))
518         self.create_enemies((prev_floor_rct.right, prev_floor_center_y,
519                              prev_floor_rct.right, prev_floor_center_y))
520
521     def create_ceil(self, ceil_center: tuple[int, int], floor_center: tuple[int, int]) >>>
522     """
523     天井を生成する関数
524     ceil_center: 天井の中心座標
525     """
526     global WIDTH, dynamic_rect_lst
527     self.blocks.add(Block(ceil_center, (WIDTH, self.__right_floor_center_y)))
528     self.__ceil_rct = self.blocks.sprites()[0].rect
529     dynamic_rect_lst.append(self.__ceil_rct)
530
531     def create_floor(self, floor_center: tuple[int, int], floor_size: tuple[int, int]) >>>
532     """
533     床を生成する関数
534     floor_center: 床の中心座標
535     floor_size: 床のサイズ
536     """
537     global WIDTH, dynamic_rect_lst
538     self.blocks.add(Block(floor_center, floor_size))
539     dynamic_rect_lst.append(self.blocks.sprites()[0].rect)
540
541     def create_obstacles(self, rangex: tuple[int, int], rangey: tuple[int, int]) >>>
542     """
543     障害物を生成する関数
544     rangex: x方向の生成範囲
545     rangey: y方向の範囲
546     """
547     for i in range(random.randint(self.min_obstacle, self.max_obstacle)):
548         self.blocks.add(Block((random.randint(*rangex), random.randint(*rangey)),
549                               (random.randint(*rangex), random.randint(*rangey))))
550         dynamic_rect_lst.append(self.blocks.sprites()[0].rect)
551
552     def create_enemies(self, rangex: tuple[int, int], rangey: tuple[int, int]) >>>
553     """
554     敵を生成する関数
555     rangex: x方向の生成範囲
556     rangey: y方向の範囲
557     """
558     for i in range(random.randint(self.min_enemy, self.max_enemy)):
559         self.enemies.add Enemy((random.randint(*rangex), random.randint(*rangey)))
560
561     class Score:
562     """
563     時間経過で増えていくスコアと
564     プレイヤー死亡時の最終スコアの表示
565     """
566     def __init__(self):

```

```

565         self.score = 0
566         self.kill_enemy = 0
567         self.progress = 0
568         self.time = 0
569         self.player_init_pos_x = 0
570         self.final_score = 0
571         self.font = pg.font.Font(None, 36)
572         self.game_over_font = pg.font.Font(None, 50)
573
574     def modify(self):
575         self.score = self.kill_enemy * 100 + self.pro
576     def increase(self, points):
577         self.time += points
578
579     def render(self, surface, pos):
580         self.modify()
581         #print(self.progress)
582         score_surface = self.font.render("Score: " +
583         surface.blit(score_surface, pos)
584
585     def render_final(self, surface):
586         self.modify()
587         final_score_surface = self.font.render(f"Game
588         restart_surface = self.font.render("Restart:
589         surface.blit(final_score_surface, (WIDTH / 2,
590         # surface.blit(restart_surface, (WIDTH / 2, H
591         # restart_surface.blit(surface, (WIDTH / 2, H
592         pg.display.update()
593
594
595     def main():
596         """
597         ゲームループ
598         """
599         global dynamic_rect_lst
600         pg.display.set_caption("ハコツミツミ(仮称)")
601         screen = pg.display.set_mode((WIDTH, HEIGHT))
602
603         bg_img = pg.Surface((WIDTH, HEIGHT))
604         dynamic_rect_lst.append(bg_img.get_rect())
605
606         player = Player(VIEW_POS)
607         level = Level()
608         score = Score()
609         score.player_init_pos_x = level.blocks.sprites()[
610
611         tmr = 0
612         clock = pg.time.Clock()
613         while True:
614             for event in pg.event.get():
615                 if event.type == pg.QUIT:
616                     return
617                 if event.type == pg.KEYDOWN and event.key
618                     # 右シフトキーが押されたら
619                     player.change_state("hyper", 400)
620                 if level.blocks.sprites()[0].rect.bottom < -H
621                     print(level.blocks.sprites()[0].rect.bott
622                     score.render_final(screen)

```

```
623         pg.time.delay(3000)
624         return
625
626     key_lst = pg.key.get_pressed()
627
628     # 各スプライトの更新
629     player.update(key_lst)
630     # Box
631     Box.bboxes.update()
632     # Bomb
633     Bomb.bombs.update()
634     # Explode
635     Explode.explodes.update()
636     # predict
637     Throw_predict.predicts.update()
638     # Enemy
639     level.enemies.update()
640     # Level
641     level.update()
642
643     # スクロール処理
644     # player以外のrectをplayerの速度に応じて移動
645     # 床はy方向のみ移動
646     for r in dynamic_rect_lst:
647         r.x -= int(player.vel[0])
648         if not player.is_grounded:
649             r.y -= int(player.vel[1])
650
651
652     #毎フレーム落下するとして初期化
653     for i in Box.bboxes:
654         i.is_ground = False
655     for i in Bomb.bombs:
656         i.is_ground = False
657
658     #Boxの接地判定
659     collide_lst_n = pg.sprite.groupcollide(Box.bboxes, Bomb.bombs, False, False)
660     for box, collide_lst in collide_lst_n.items():
661         if len(collide_lst) == 0:
662             box.is_ground = False
663         for b in collide_lst:
664             # x方向
665             if box.rect.right <= b.rect.left + b.rect.width:
666                 if box.vel[0] < 0:
667                     gap = b.rect.right - box.rect.right
668                     box.rect.centerx = box.rect.centerx - gap
669                     box.vel[0] = 0
670                 elif box.vel[0] > 0:
671                     gap = box.rect.right - b.rect.left
672                     box.rect.centerx = box.rect.centerx + gap
673                     box.vel[0] = 0
674             # y方向
675             else:
676                 if box.vel[1] > 0:
677                     gap = box.rect.bottom - b.rect.top
678                     box.rect.centery = box.rect.centery - gap
679                     box.is_ground = True
680                     box.vel[1] = 0
```

```
681         box.is_ground = True
682     elif box.vel[1] < 0:
683         gap = b.rect.bottom - box.rect.bottom
684         box.rect.centery = box.rect.bottom + gap
685         box.vel[1] = 0
686
687     # Boxの摩擦処理
688     if box.is_ground:
689         box.vel[0] = (0.3 * box.vel[0])
690
691     # Bombの接地判定
692     collide_lst = pg.sprite.groupcollide(Bomb.bomb, Box, False, False)
693     for i in collide_lst:
694         i.is_ground = True
695
696     # Box同士の衝突判定
697     collide_lst = pg.sprite.groupcollide(Box, Box, False, False)
698
699     for obj, collide_lst_2 in collide_lst.items():
700         if len(collide_lst_2) > 1:
701             for obj2 in collide_lst_2:
702                 if not obj is obj2:
703
704                     # y軸
705                     if obj.rect.centery < obj2.rect.centery:
706                         obj.is_ground = True
707                         obj.rect.centery -= (obj2.rect.centery - obj.rect.centery)
708                         obj.vel[1] = 0
709                         obj.vel[0] = 0
710                         break
711                     else:
712                         pass
713
714                     # x軸方向の当たり判定
715                     # print(id(obj), obj.is_ground)
716                     if not obj.is_ground:
717                         if obj2.rect.centerx > obj.rect.centerx:
718                             obj.rect.centerx -= (obj2.rect.centerx - obj.rect.centerx)
719                             obj.vel[0] = 0
720                         elif obj2.rect.left < obj2.rect.right:
721                             obj.rect.centerx += (obj2.rect.left - obj2.rect.right)
722                             obj.vel[0] = 0
723
724
725     # BombとBoxのCollide
726     collide_lst = pg.sprite.groupcollide(Bomb.bomb, Box, False, False)
727     for bomb in collide_lst:
728         bomb.set_vel(0, 0)
729         bomb.is_ground = True
730
731     # Bombによって召喚されたExplodeとBoxのCollide
732     collide_lst = pg.sprite.groupcollide(Explode, Box, False, False)
733     for key, items in collide_lst.items():
734         for item in items:
735             throw_arg = [0, 0]
736             item_pos = list(item.rect.center)
737             key_pos = list(key.rect.center)
738             power_border = 0.5
739             throw_arg[0] = -(key_pos[0] - item_pos[0])
```

```
739         throw_arg[1] = -(key_pos[1] - item_pos[1])
740         item.vel[0] += throw_arg[0]
741         item.vel[1] += throw_arg[1]
742
743
744
745     #予測線の接地判定
746     collide_lst = pg.sprite.groupcollide(Throw_projectile, player, False, False)
747
748
749     # ブロックとの衝突判定
750     collide_lst = pg.sprite.spritecollide(player, dynamic_rect_lst, False)
751     if len(collide_lst) == 0:
752         player.is_grounded = False
753     else:
754         for b in collide_lst:
755             # x方向
756             if player.rect.right <= b.rect.left + 1:
757                 if player.vel[0] < 0:
758                     gap = b.rect.right - player.rect.left
759                     for r in dynamic_rect_lst:
760                         r.x -= gap
761                     player.set_vel(0)
762                 elif player.vel[0] > 0:
763                     gap = player.rect.right - b.rect.left
764                     for r in dynamic_rect_lst:
765                         r.x += gap
766                     player.set_vel(0)
767
768             # y方向
769             else:
770                 if player.vel[1] > 0:
771                     gap = player.rect.bottom - b.rect.top
772                     for r in dynamic_rect_lst:
773                         r.y += gap
774                     player.is_grounded = True
775                 elif player.vel[1] < 0:
776                     gap = b.rect.bottom - player.rect.top
777                     for r in dynamic_rect_lst:
778                         r.y -= gap
779                     player.set_vel(vy=0)
780
781     #ExplodeとPlayerの当たり判定 あたると吹っ飛ぶ
782     collide_lst = pg.sprite.spritecollide(player, explode_lst, False)
783     if player.state != "hyper":
784         for explode in collide_lst:
785             throw_arg = [0,0]
786             explode_pos = list(explode.rect.center)
787             player_pos = list(player.rect.center)
788             power_border = 3
789             throw_arg[0] = -(explode_pos[0] - player_pos[0])
790             throw_arg[1] = -(explode_pos[1] - player_pos[1])
791
792             player.add_vel(throw_arg[0], throw_arg[1])
793
794
795
796     #BoxにPlayerが乗るための接地判定
```



```
797         collide_lst = pg.sprite.spritecollide(player,
798         for b in collide_lst:
799             # x方向
800             if False:
801                 pass
802             # y方向
803             else:
804                 if player.vel[1] > 0 and b.rect.cente
805
806                 gap = player.rect.bottom - b.rect
807                 for r in dynamic_rect_lst:
808                     r.y += gap
809                 player.is_grounded = True
810
811                 player.set_vel(vy=0)
812         # Playerの摩擦処理
813         if (player.is_grounded):
814             if player.vel[0] == 0:
815                 pass
816             elif abs(player.vel[0]) < 0.001:
817                 player.set_vel(0)
818             else:
819                 player.set_vel(0.7 * player.vel[0])
820
821         # Enemyの当たり判定
822         score.kill_enemy += len(pg.sprite.spritecollide
823
824         # 各種描画処理
825         screen.blit(bg_img, (0, 0))
826         level.blocks.draw(screen)
827         level.enemies.draw(screen)
828         Box.bboxes.draw(screen)
829         Bomb.bombs.draw((screen))
830         Explode.explodes.draw((screen))
831         Throw_predict.predicts.draw((screen))
832         screen.blit(player.image, player.rect)
833         score.render(screen, (WIDTH - 150, 10))
834         pg.display.update()
835
836         tmr += 1
837         score.progress = int(max(score.progress, abs(s
838         if tmr % 60 == 0:
839             score.increase(1)
840         clock.tick(60)
841
842     if __name__ == "__main__":
843         pg.init()
844         main()
845         pg.quit()
846         sys.exit()
847
```