c0b21082 / ProjExD_05-1 Public

forked from omochio/ProjExD_05

Code Pull requests Actions Projects Wiki Security Insights Settings

main ProjExD_05-1 / README.md Go to file t ···

omochio Merge branch 'main' into main 10 minutes ago

72 lines (49 loc) · 1.48 KB

Preview Code Blame Raw

ハコツミツミ

実行環境の必要条件

- python >= 3.10
- pygame >= 2.1

ゲームの概要

ハコを積み上げてブロックや穴を越え, 攻撃を躱し, 敵を倒し, 時には自分も爆破し, 進み続けるアクションゲーム

操作方法

- キーボード
    - A or Light: 左移動
    - D or Right: 右移動
    - Space or Up: ジャンプ
    - Right Shift: 無敵
    - Right Ctrl: 予測線表示
- マウス
    - Left Click: ハコ発射
    - Right Click: ボム発射

スコア

以下のことでスコア上昇！

- 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

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

WIDTH = 1600
HEIGHT = 1000
# ビューの座標
VIEW_POS = (WIDTH // 2, HEIGHT - 200)

# スクロールのために動的に変更されるrectのリスト
dynamic_rect_lst = []

class Player(pg.sprite.Sprite):
    """
    Playerに関するクラス
    """
    # 入力と移動方向の対応
    __move_dict = {
        pg.K_LEFT: (-1, 0),
        pg.K_a: (-1, 0),
        pg.K_RIGHT: (1, 0),
        pg.K_d: (1, 0),
        pg.K_UP: (0, -1),
        pg.K_SPACE: (0, -1)
    }

    def __init__(self, center: tuple[int, int]):
        """
        Playerクラスの初期化
        center: 初期座標
        """
        super().__init__()
        self.__size = (64, 64)
        self.image = pg.Surface(self.__size)
        self.image.fill((255, 255, 255))
        self.rect = self.image.get_rect()
        self.rect.center = center
        self.my_timer = 0
        self.box_timer = 0
        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__

```
        self.is_pre_predict = False
        self.__acc = [.0, .0]
        self.__vel = [.0, .0]
        self.__gravity_acc = 1
        self.__walk_acc = 2
        self.__walk_vel_max = 10
        self.__jump_init_vel = 20
        self.__is_grounded = False
        self.state = "normal" # プレイヤーの状態
        self.hyper_life = 0 # 残りの無敵状態時間

    @property
    def is_grounded(self) -> bool:
        """
        接地判定変数のgetter
        返り値: 接地判定変数の値
        """
        return self.__is_grounded

    @is_grounded.setter
    def is_grounded(self, value: bool):
        """
        接地判定変数のsetter
        value: 接地判定変数の値
        """
        self.__is_grounded = value

    @property
    def vel(self) -> list[float, float]:
        """
        速度のgetter
        返り値: 速度のリスト
        """
        return self.__vel.copy()

    def set_vel(self, vx: float = None, vy: float = N
        """
        速度のsetter
        Noneを入れた方向は変更しない
        vx: x方向の速度
        vy: y方向の速度
        """
        if vx is not None:
            self.__vel[0] = vx
        if vy is not None:
            self.__vel[1] = vy

    def add_vel(self, vx: float = .0, vy: float = .0)
        """
        速度の加算
        vx: x方向の加算速度
        vy: y方向の加算速度
        """
        self.__vel[0] += vx
        self.__vel[1] += vy

    def change_state(self, state: str, hyper_life: ir
        """
```

func update
func set_vel
func is_moving
class Bomb
func __init__
func update
func set_vel

```
        右シフトキーが押された時に，プレイヤーを無敵状態に
        引数1 state : プレイヤーの状態
        引数2 hyper_life : 無敵状態になっている時間
        戻り値 : なし
        """
        self.state = state
        self.hyper_life = hyper_life

    def check_hyper(self):
        """
        プレイヤーが無敵状態かどうかを判定し，プレイヤーの
        戻り値 : なし
        """
        if self.state == "hyper":
            # プレイヤーが無敵状態だったら
            self.image.fill((168, 88, 168)) # プレイヤ
            self.hyper_life += -1 # 残りの無敵状態時間

        if self.hyper_life < 0: # 残りの無敵状態時間が0
            self.state == "normal" # プレイヤーを通常状
            self.image.fill((255, 255, 255)) # プレイ


    def update(self, key_lst: dict):
        """
        Playerの更新を行う
        key_lst: 押されているキーのリスト
        """

        self.my_timer += 1
        self.update_box(key_lst)
        self.update_bomb(key_lst)
        self.update_throw_predict(key_lst)
        self.__acc = [.0, .0]
        # 入力と移動方向dictに応じて加速度を設定
        for d in __class__.__move_dict:
            if key_lst[d]:
                self.__acc[0] += self.__walk_acc * __
                # 接地時のみジャンプ可能
                if self.is_grounded:
                    self.set_vel(vy=self.__jump_init_
                    if self.vel[1] < 0:
                        self.is_grounded = False

        # 重力加速度を加算
        if not self.is_grounded:
            self.__acc[1] += self.__gravity_acc

        # 加速度と速度上限から速度を計算
        self.add_vel(self.__acc[0])
        if self.vel[0] < -self.__walk_vel_max:
            self.set_vel(-self.__walk_vel_max)
        elif self.vel[0] > self.__walk_vel_max:
            self.set_vel(self.__walk_vel_max)
        self.add_vel(vy=self.__acc[1])

        self.check_hyper()

```

```python

    def update_box(self,key_lst: dict):
        """
        Press mouse Left
        box throw
        """

        #次に投げれるようになるまでのフレーム数
        if self.my_timer - self.box_timer < 10:
            return


        pg.event.get()
        if pg.mouse.get_pressed()[0]:
            self.box_timer = self.my_timer
            throw_arg = [0,0]
            mouse_pos = list(pg.mouse.get_pos())
            player_pos = list(self.rect.center)
            throw_arg[0] = (mouse_pos[0] - player_pos
            throw_arg[1] = (mouse_pos[1] - player_pos
            Box((self.rect.centerx + throw_arg[0],sel



    def update_bomb(self,key_lst: dict):
        """
        Press mouse Riglt
        bomb throw
        """

        #次に投げれるようになるまでのフレーム数
        if self.my_timer - self.box_timer < 30:
            return


        pg.event.get()
        if pg.mouse.get_pressed()[2]:
            self.box_timer = self.my_timer
            throw_arg = [0,0]
            mouse_pos = list(pg.mouse.get_pos())
            player_pos = list(self.rect.center)
            throw_arg[0] = (mouse_pos[0] - player_pos
            throw_arg[1] = (mouse_pos[1] - player_pos
            Bomb(self.rect.center,tuple(throw_arg),pc

    def update_throw_predict(self,key_lst: dict):
        """
        Press Shift
        draw throw curve
        """



        pg.event.get()
        #CTRLで予測線
        if (key_lst[pg.K_RCTRL]):
            if not self.is_pre_predict:
                self.is_predict = not self.is_predict
                self.is_pre_predict = True
        else:
            self.is_pre_predict = False

        #次に投げれるようになるまでのフレーム数
        if self.my_timer - self.curve_timer < 10:
            return

        if self.is_predict:
            self.curve_timer = self.my_timer
            throw_arg = [0,0]
            mouse_pos = list(pg.mouse.get_pos())
            player_pos = list(self.rect.center)
            throw_arg[0] = (mouse_pos[0] - player_pos
            throw_arg[1] = (mouse_pos[1] - player_pos
            Throw_predict(self.rect.center,tuple(thro

class Block(pg.sprite.Sprite):
    """
    初期生成されるブロックに関するクラス
    """
    def __init__(self, center: tuple[int, int], size:
        super().__init__()
        self.__size = size
        self.image = pg.Surface(size)
        self.image.fill((127, 127, 127))
        self.rect = self.image.get_rect()
        self.rect.center = center

    @property
    def size(self) -> tuple[int, int]:
        """
        サイズのgetter
        返り値: サイズのタプル
        """
        return self.__size

class Box(pg.sprite.Sprite):
    """
    playerがなげるBoxClassです
    """
    boxes = pg.sprite.Group()
    def __init__(self, pos: tuple[int, int],vel:tuple
        global dynamic_rect_lst
        super().__init__()
        self.image = pg.Surface((50, 50))
        self.image.fill((0, 255, 255))
        self.rect = self.image.get_rect()
        self.rect.center = pos
        self.gravity_val = 1
        self.life = 0
        self.is_ground = False
        self.vel = list(vel)
        self.acc = [0,0]
        self.acc = [0,self.gravity_val]
        __class__.boxes.add(self)
        dynamic_rect_lst.append(self.rect)


    def update(self):

        self.life += 1
        if self.life > 6000:
            self.kill()
        self.vel[0] += self.acc[0]
        self.vel[1] += self.acc[1]



        if self.is_ground:
            self.vel[1] = 0
            self.vel[0] = 0

        self.rect.x += self.vel[0]
        self.rect.y += self.vel[1]

    def set_vel(self,vx,vy):
        self.vel[1] = vy
        self.vel[0] = vx

    def is_moving(self):
        #[0,0]でないならFalse
        return not self.vel == [0,0]

class Bomb(pg.sprite.Sprite):
    """
    playerがなげるBombClassです
    """
    bombs = pg.sprite.Group()
    def __init__(self, pos: tuple[int, int],vel:tuple
        global dynamic_rect_lst
        super().__init__()
        self.image = pg.Surface((30, 30))
        self.image.fill((255, 128, 0))
        self.rect = self.image.get_rect()
        #self.image.set_alpha(128)
        self.rect.center = pos
        self.gravity_val = 1
        self.life = 0
        self.is_ground = False
        self.vel = list(vel)
        self.acc = [0,0]
        self.acc = [0,self.gravity_val]
        __class__.bombs.add(self)
        dynamic_rect_lst.append(self.rect)

    def update(self):
        life_max = 180
        self.life += 1

        #自動で消えるまでの時間
        if self.life >= life_max:
            Explode(self.rect.center)
            self.kill()

        #爆発までの時間を色で表現
        self.image.fill((255 - 128*int((self.life/lif
```

```python

        self.vel[0] += self.acc[0]
        self.vel[1] += self.acc[1]



        if self.is_ground:
            self.vel[1] = 0
            self.vel[0] = 0

        self.rect.x += self.vel[0]
        self.rect.y += self.vel[1]

    def set_vel(self,vx,vy):
        self.vel[1] = vy
        self.vel[0] = vx

class Explode(pg.sprite.Sprite):
    """
    Bombが爆発した時に呼び出されるExplodeClassです
    """
    explodes = pg.sprite.Group()
    def __init__(self, pos: tuple[int, int],power:flo
        global dynamic_rect_lst
        super().__init__()
        rad = power * 16
        self.image = pg.Surface((rad, rad))
        self.image.fill((200, 0, 0))
        pg.draw.circle(self.image, (200, 0, 0), (rad,
        self.image.set_colorkey((255, 255, 255))
        self.image.set_alpha(128)
        self.rect = self.image.get_rect()
        self.rect.center = pos
        self.life = 0
        __class__.explodes.add(self)
        dynamic_rect_lst.append(self.rect)

    def update(self):
        self.life += 1
        #自動で消えるまでの時間
        if self.life > 12:
            self.kill()

class Throw_predict(pg.sprite.Sprite):
    """
    playerがなげるものの予測線Classです
    """
    predicts = pg.sprite.Group()
    def __init__(self, pos: tuple[int, int],vel:tuple
        global dynamic_rect_lst
        super().__init__()
        self.image = pg.Surface((15, 15))
        self.image.fill((255, 200, 255))
        self.rect = self.image.get_rect()
        self.rect.center = pos
        self.gravity_val = 1
        self.life = 0
        self.vel = list(vel)
        self.acc = [0,0]
        self.acc = [0,self.gravity_val]
        __class__.predicts.add(self)
        dynamic_rect_lst.append(self.rect)

    def update(self):

        self.life += 1
        #自動で消えるまでの時間
        if self.life > 20:
            self.kill()
        self.vel[0] += self.acc[0]
        self.vel[1] += self.acc[1]

        self.rect.x += self.vel[0]
        self.rect.y += self.vel[1]

    def set_vel(self,vx,vy):
        self.vel[1] = vy
        self.vel[0] = vx

class Enemy(pg.sprite.Sprite):  # エネミークラス
    x = 400
    y = 700
    def __init__(self, center: tuple[int, int]):
        global dynamic_rect_lst
        super().__init__()
        self.image = pg.Surface((64, 64))
        self.image.fill((255, 0, 0))
        self.rect = self.image.get_rect()
        dynamic_rect_lst.append(self.rect)
        self.rect.center = center
        self.life = 0

    def update(self):
        global VIEW_POS
        if self.life % 60 == 0 and 0 <= self.rect.cer
            self.throw_bomb()
        self.life += 1

    def throw_bomb(self):
        throw_arg = [0,0]
        player_pos = list(VIEW_POS)
        enemy_pos = list(self.rect.center)
        throw_arg[0] = (player_pos[0] - enemy_pos[0])
        throw_arg[1] = (player_pos[1] - enemy_pos[1])
        Bomb(self.rect.center,tuple(throw_arg),power=

class Level():
    """
    レベル生成と保持を担うクラス
    """
    def __init__(self):
        global dynamic_rect_lst
        self.blocks = pg.sprite.Group()
        self.__flcl_height = 100    # 床と天井の高さ
        self.__ceil_y = -HEIGHT // 2    # 天井の中心y
        # 床
        self.min_floor_width = 100
        self.min_floor_width = 100
        self.max_floor_width = WIDTH // 2
        # 天井の生成
        self.create_ceil((WIDTH // 2, self.__ceil_y))
        # 床の生成
        self.blocks.add(Block((WIDTH // 2, HEIGHT), (
        dynamic_rect_lst.append(self.blocks.sprites()
        self.__left_floor_rct = self.blocks.sprites()
        self.__right_floor_rct = self.blocks.sprites(

        # 障害物
        self.min_obstacle_count = 50
        self.max_obstacle_count = 100
        self.min_obstacle_width = 50
        self.min_obstacle_height = 50
        self.max_obstacle_width = 100
        self.max_obstacle_height = 100

        # 穴
        self.min_hole_width = 0
        self.max_hole_width = WIDTH // 2

        # 敵
        self.enemies = pg.sprite.Group()
        self.min_enemy_count = 10
        self.max_enemy_count = 20

    def update(self):
        """
        レベルの更新を行う
        """
        global WIDTH
        # 左端の床のx座標が-WIDHT//2より大きくなったら生
        if self.__left_floor_rct.left >= -WIDTH // 2:
            self.create_ceil((self.__left_floor_rct.l
            prev_floor_rct = self.__left_floor_rct
            total = 0
            # 生成した床の長さが穴を含めてWIDTHを超えるま
            while total < WIDTH:
                offset = random.randint(self.min_hole
                sizex = random.randint(self.min_floor
                if total + offset + sizex >= WIDTH:
                    sizex = WIDTH - total
                    offset = 0
                    total += sizex
                else:
                    total += offset + sizex
                self.create_floor((self.__left_floor_
                self.__left_floor_rct = self.blocks.s
            self.create_obstacles((self.__left_floor_
            self.create_enemies((self.__left_floor_rc
        # 右端の床のx座標がWIDHT * 3//2より小さくなったら
        elif self.__right_floor_rct.right <= WIDTH *
            self.create_ceil((self.__right_floor_rct.
            prev_floor_rct = self.__right_floor_rct
            total = 0
            # 生成した床の長さが穴を含めてWIDTHを超えるま
            while total < WIDTH:
                offset = random.randint(self.min_hole
                offset = random.randint(self.min_hol
                sizex = random.randint(self.min_floor
                if total + offset + sizex >= WIDTH:
                    sizex = WIDTH - total
                    offset = 0
                    total += sizex
                else:
                    total += offset + sizex
                self.create_floor((self.__right_floor
                self.__right_floor_rct = self.blocks.
            self.create_obstacles((prev_floor_rct.rig
            self.create_enemies((prev_floor_rct.right

    def create_ceil(self, ceil_center: tuple[int, int
        """
        天井を生成する関数
        ceil_center: 天井の中心座標
        """
        global WIDTH, dynamic_rect_lst
        self.blocks.add(Block(ceil_center, (WIDTH, se
        self.__ceil_rct = self.blocks.sprites()[-1].r
        dynamic_rect_lst.append(self.__ceil_rct)


    def create_floor(self, floor_center: tuple[int, i
        """
        床を生成する関数
        floor_center: 床の中心座標
        floor_size: 床のサイズ
        """
        global WIDTH, dynamic_rect_lst
        self.blocks.add(Block(floor_center, floor_siz
        dynamic_rect_lst.append(self.blocks.sprites()

    def create_obstacles(self, rangex: tuple[int, int
        """
        障害物を生成する関数
        rangex: x方向の生成範囲
        rangey: y方向の範囲
        """
        for i in range(random.randint(self.min_obstac
            self.blocks.add(Block((random.randint(*ra
            dynamic_rect_lst.append(self.blocks.sprit

    def create_enemies(self, rangex: tuple[int, int],
        """
        敵を生成する関数
        rangex: x方向の生成範囲
        rangey: y方向の範囲
        """
        for i in range(random.randint(self.min_enemy_
            self.enemies.add(Enemy((random.randint(*r

class Score:
    """
    時間経過で増えていくスコアと
    プレイヤー死亡時の最終スコアの表示
    """
    def __init__(self):
        self.score = 0
        self.kill_enemy = 0
        self.progress = 0
        self.time = 0
        self.player_init_pos_x = 0
        self.final_score = 0
        self.font = pg.font.Font(None, 36)
        self.game_over_font = pg.font.Font(None, 50)

    def modify(self):
        self.score = self.kill_enemy * 100 + self.pr
    def increase(self, points):
        self.time += points

    def render(self, surface, pos):
        self.modify()
        #print(self.progress)
        score_surface = self.font.render("Score: " +
        surface.blit(score_surface, pos)

    def render_final(self,surface):
        self.modify()
        final_score_surface = self.font.render(f"Game
        restart_surface = self.font.render("Restart:
        surface.blit(final_score_surface, (WIDTH / 2,
        # surface.blit(restart_surface, (WIDTH / 2, H
        # restart_surface.blit(surface, (WIDTH / 2, H
        pg.display.update()


def main():
    """
    ゲームループ
    """
    global dynamic_rect_lst
    pg.display.set_caption("ハコツミツミ(仮称)")
    screen = pg.display.set_mode((WIDTH, HEIGHT))

    bg_img = pg.Surface((WIDTH, HEIGHT))
    dynamic_rect_lst.append(bg_img.get_rect())

    player = Player(VIEW_POS)
    level = Level()
    score = Score()
    score.player_init_pos_x = level.blocks.sprites()[

    tmr = 0
    clock = pg.time.Clock()
    while True:
        for event in pg.event.get():
            if event.type == pg.QUIT:
                return
            if event.type == pg.KEYDOWN and event.key
                # 右シフトキーが押されたら
                player.change_state("hyper", 400)
        if level.blocks.sprites()[0].rect.bottom < -H
            print(level.blocks.sprites()[0].rect.bott
            score.render_final(screen)
            pg.time.delay(3000)
            return

        key_lst = pg.key.get_pressed()

        # 各スプライトの更新
        player.update(key_lst)
        # Box
        Box.boxes.update()
        # Bomb
        Bomb.bombs.update()
        # Explode
        Explode.explodes.update()
        # predict
        Throw_predict.predicts.update()
        # Enemy
        level.enemies.update()
        # Level
        level.update()

        # スクロール処理
        # player以外のrectをplayerの速度に応じて移動
        # 床はy方向のみ移動
        for r in dynamic_rect_lst:
            r.x -= int(player.vel[0])
            if not player.is_grounded:
                r.y -= int(player.vel[1])


        #毎フレーム落下するとして初期化
        for i in Box.boxes:
            i.is_ground = False
        for i in Bomb.bombs:
            i.is_ground = False

        #Boxの接地判定
        collide_lst_n = pg.sprite.groupcollide(Box.bo
        for box,collide_lst in collide_lst_n.items():
            if len(collide_lst) == 0:
                box.is_ground = False
            for b in collide_lst:
                # x方向
                if  box.rect.right <= b.rect.left + b
                    if box.vel[0] < 0:
                        gap = b.rect.right - box.rect
                        box.rect.centerx = box.rect.c
                        box.vel[0] = 0
                    elif box.vel[0] > 0:
                        gap = box.rect.right - b.rect
                        box.rect.centerx = box.rect.c

                        box.vel[0] = 0
                # y方向
                else:
                    if box.vel[1] > 0:

                        gap = box.rect.bottom - b.rec
                        box.rect.centery =  box.rect.
                        box.is_ground = True
                    elif box.vel[1] < 0:
                        gap = b.rect.bottom - box.rec
                        box.rect.centery = box.rect.c
                    box.vel[1] = 0

            # Boxの摩擦処理
            if box.is_ground:
                box.vel[0] = (0.3 * box.vel[0])

        #Bombの接地判定
        collide_lst = pg.sprite.groupcollide(Bomb.bom
        for i in collide_lst:
            i.is_ground = True

        #Box同士の衝突判定
        collide_lst = pg.sprite.groupcollide(Box.boxe

        for obj,collide_lst_2 in collide_lst.items():
            if len(collide_lst_2) > 1:
                for obj2 in collide_lst_2:
                    if not obj is obj2:

                        #y軸
                        if obj.rect.centery < obj2.re
                            obj.is_ground = True
                            obj.rect.centery -= (obj.
                            obj.vel[1] = 0
                            obj.vel[0] = 0
                            break
                        else:
                            pass

                        #x軸方向の当たり判定
                        #print(id(obj),obj.is_ground)
                        if not obj.is_ground:
                            if obj2.rect.centerx > ob
                                obj.rect.centerx -= (
                                obj.vel[0] = 0
                            elif obj.rect.left < obj2
                                obj.rect.centerx += (
                                obj.vel[0] = 0


        #BombとBoxのCollide
        collide_lst = pg.sprite.groupcollide(Bomb.bom
        for bomb in collide_lst:
            bomb.set_vel(0,0)
            bomb.is_ground = True
        #Bombによって召喚されたExplodeとBoxのCollide
        collide_lst = pg.sprite.groupcollide(Explode.
        for key,items in collide_lst.items():
            for item in items:
                throw_arg = [0,0]
                item_pos = list(item.rect.center)
                key_pos = list(key.rect.center)
                power_border = 0.5
                throw_arg[0] = -(key_pos[0] - item_pc
                throw_arg[1] = -(key_pos[1] - item_po
                item.vel[0] += throw_arg[0]
                item.vel[1] += throw_arg[1]



        #予測線の接地判定
        collide_lst = pg.sprite.groupcollide(Throw_pr


        # ブロックとの衝突判定
        collide_lst = pg.sprite.spritecollide(player,
        if len(collide_lst) == 0:
            player.is_grounded = False
        else:
            for b in collide_lst:
                # x方向
                if player.rect.right <= b.rect.left +
                    if player.vel[0] < 0:
                        gap = b.rect.right - player.r
                        for r in dynamic_rect_lst:
                            r.x -= gap
                        player.set_vel(0)
                    elif player.vel[0] > 0:
                        gap = player.rect.right - b.r
                        for r in dynamic_rect_lst:
                            r.x += gap
                        player.set_vel(0)

                # y方向
                else:
                    if player.vel[1] > 0:
                        gap = player.rect.bottom - b.
                        for r in dynamic_rect_lst:
                            r.y += gap
                        player.is_grounded = True
                    elif player.vel[1] < 0:
                        gap = b.rect.bottom - player.
                        for r in dynamic_rect_lst:
                            r.y -= gap
                    player.set_vel(vy=0)

        #ExplodeとPlayerの当たり判定 あたると吹っ飛ぶ
        collide_lst = pg.sprite.spritecollide(player,
        if player.state != "hyper":
            for explode in collide_lst:
                throw_arg = [0,0]
                explode_pos = list(explode.rect.cente
                player_pos = list(player.rect.center)
                power_border = 3
                throw_arg[0] = -(explode_pos[0] - pla
                throw_arg[1] = -(explode_pos[1] - pla

                player.add_vel(throw_arg[0],throw_arg



        #BoxにPlayerが乗るための接地判定
        collide_lst = pg.sprite.spritecollide(player,
        for b in collide_lst:
            # x方向
            if False:
                pass
            # y方向
            else:
                if player.vel[1] > 0 and b.rect.cente

                    gap = player.rect.bottom - b.rect
                    for r in dynamic_rect_lst:
                        r.y += gap
                    player.is_grounded = True

                    player.set_vel(vy=0)
        # Playerの摩擦処理
        if (player.is_grounded):
            if player.vel[0] == 0:
                pass
            elif abs(player.vel[0]) < 0.001:
                player.set_vel(0)
            else:
                player.set_vel(0.7 * player.vel[0])

        # Enemyの当たり判定
        score.kill_enemy += len(pg.sprite.spritecolli

        # 各種描画処理
        screen.blit(bg_img, (0, 0))
        level.blocks.draw(screen)
        level.enemies.draw(screen)
        Box.boxes.draw(screen)
        Bomb.bombs.draw((screen))
        Explode.explodes.draw((screen))
        Throw_predict.predicts.draw((screen))
        screen.blit(player.image, player.rect)
        score.render(screen, (WIDTH - 150, 10))
        pg.display.update()

        tmr += 1
        score.progress = int(max(score.progress,abs(s
        if tmr % 60 == 0:
            score.increase(1)
        clock.tick(60)

if __name__ == "__main__":
    pg.init()
    main()
    pg.quit()
    sys.exit()

```