ダークこうかとんを倒せ！

実行環境の必要条件

- python >= 3.10
- pygame >= 2.1

ゲームの概要

ボスであるダークこうかとんをこうかとんが倒す。

ゲームの実装

共通基本機能

- 背景画像の横スクロールと主人公キャラクターの描画とex4の追加機能以外

担当追加機能

- ボス追加（担当：あだち）：ボスのダークこうかとんを召喚、当たり判定を決める
- ボス背景（担当：なかやま）：敵の撃破数で背景変化　＋　ボスステージの時のみBGM再生
- チャージショット（担当：かねこ）：こうかとんがチャージショットを打つための関数
- ライフ（担当:くらた）：こうかとんと敵のＨＰを決める
- ボスの攻撃パターン（担当：ほりうち）：ボスの攻撃パターンを追加する

ToDo

-
-

メモ

-
-

ProjExD_Group02 / darkkokaton.py

c0a2311023 ボス背景とＢＧＭ追加 16 minutes ago

402 lines (338 loc) · 13.9 KB

```python
import math
import os
import random
import sys
import time
import pygame as pg


WIDTH, HEIGHT = 1600, 900  # ゲームウィンドウの幅，高さ
os.chdir(os.path.dirname(os.path.abspath(__file__)))


def check_bound(obj_rct:pg.Rect) -> tuple[bool, bool]:
    """
    Rectの画面内外判定用の関数
    引数：こうかとんRect，または，爆弾Rect，またはビームRect
    戻り値：横方向判定結果，縦方向判定結果（True：画面内／False：画面外）
    """
    yoko, tate = True, True
    if obj_rct.left < 0 or WIDTH < obj_rct.right:  # 横方向のはみ出し判定
        yoko = False
    if obj_rct.top < 0 or HEIGHT < obj_rct.bottom:  # 縦方向のはみ出し判定
        tate = False
    return yoko, tate


def calc_orientation(org: pg.Rect, dst: pg.Rect) -> tuple[float, float]:
    """
    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
    引数1 org：爆弾SurfaceのRect
    引数2 dst：こうかとんSurfaceのRect
    戻り値：orgから見たdstの方向ベクトルを表すタプル
    """
    x_diff, y_diff = dst.centerx-org.centerx, dst.centery-org.centery
    norm = math.sqrt(x_diff**2+y_diff**2)
    return x_diff/norm, y_diff/norm


class Bird(pg.sprite.Sprite):
    """
    ゲームキャラクター（こうかとん）に関するクラス
    """
```

```python
    delta = {  # 押下キーと移動量の辞書
        pg.K_UP: (0, -1),
        pg.K_DOWN: (0, +1),
        pg.K_LEFT: (-1, 0),
        pg.K_RIGHT: (+1, 0),
    }

    def __init__(self, num: int, xy: tuple[int, int]):
        """
        こうかとん画像Surfaceを生成する
        引数1 num：こうかとん画像ファイル名の番号
        引数2 xy：こうかとん画像の位置座標タプル
        """
        super().__init__()
        img0 = pg.transform.rotozoom(pg.image.load(f"fig/{num}.png"), 0, 2.0)
        img = pg.transform.flip(img0, True, False)  # デフォルトのこうかとん
        self.imgs = {
            (+1, 0): img,
            (+1, -1): img,
            (0, -1): img,
            (-1, -1): img,
            (-1, 0): img,
            (-1, +1): img,
            (0, +1): img,
            (+1, +1): img,
        }
        self.dire = (+1, 0)
        self.image = self.imgs[self.dire]
        self.rect = self.image.get_rect()
        self.rect.center = xy
        self.speed = 10
        self.state = "normal"
        self.hyper_life = 0



    def change_img(self, num: int, screen: pg.Surface):
        """
        こうかとん画像を切り替え，画面に転送する
        引数1 num：こうかとん画像ファイル名の番号
        引数2 screen：画面Surface
        """
        self.image = pg.transform.rotozoom(pg.image.load(f"fig/{num}.png"), 0, 2.0)
        screen.blit(self.image, self.rect)

    def update(self, key_lst: list[bool], screen: pg.Surface):
        """
        押下キーに応じてこうかとんを移動させる
        引数1 key_lst：押下キーの真理値リスト
        引数2 screen：画面Surface
        """
        sum_mv = [0, 0]
        cur_speed = self.speed
        if key_lst[pg.K_LSHIFT]:
            cur_speed = 20
        for k, mv in __class__.delta.items():
            if key_lst[k]:
                sum_mv[0] += mv[0]
```

```
                sum_mv[1] += mv[1]
            self.rect.move_ip(cur_speed*sum_mv[0], cur_speed*sum_mv[1])
            if check_bound(self.rect) != (True, True):
                self.rect.move_ip(-self.speed*sum_mv[0], -self.speed*sum_mv[1])
            if not (sum_mv[0] == 0 and sum_mv[1] == 0):
                self.dire = tuple(sum_mv)
                self.image = self.imgs[self.dire]

            if self.state == "hyper":
                self.image = pg.transform.laplacian(self.image)
                self.hyper_life -= 1
                if self.hyper_life < 0:
                    self.state = "normal"


            screen.blit(self.image, self.rect)



class Bomb(pg.sprite.Sprite):
    """
    爆弾に関するクラス
    """
    colors = [(255, 0, 0), (0, 255, 0), (0, 0, 255), (255, 255, 0), (255, 0, 255), (0, 255, 255)]

    def __init__(self, emy: "Enemy", bird: Bird):
        """
        爆弾円Surfaceを生成する
        引数1 emy：爆弾を投下する敵機
        引数2 bird：攻撃対象のこうかとん
        """
        super().__init__()
        rad = random.randint(10, 50)  # 爆弾円の半径：10以上50以下の乱数
        self.image = pg.Surface((2*rad, 2*rad))
        color = random.choice(__class__.colors)  # 爆弾円の色：クラス変数からランダム選択
        pg.draw.circle(self.image, color, (rad, rad), rad)
        self.image.set_colorkey((0, 0, 0))
        self.rect = self.image.get_rect()
        # 爆弾を投下するemyから見た攻撃対象のbirdの方向を計算
        self.vx, self.vy = calc_orientation(emy.rect, bird.rect)
        self.rect.centerx = emy.rect.centerx
        self.rect.centery = emy.rect.centery+emy.rect.height/2
        self.speed = 6

    def update(self):
        """
        爆弾を速度ベクトルself.vx, self.vyに基づき移動させる
        引数 screen：画面Surface
        """
        self.rect.move_ip(self.speed*self.vx, self.speed*self.vy)
        if check_bound(self.rect) != (True, True):
            self.kill()


class Beam(pg.sprite.Sprite):
    """
    ビームに関するクラス
    """
    def __init__(self, bird: Bird):
```

```python
        """
        ビーム画像Surfaceを生成する
        引数 bird：ビームを放つこうかとん
        """
        super().__init__()
        self.vx, self.vy = (+1, 0)
        angle = math.degrees(math.atan2(-self.vy, self.vx))
        self.image = pg.transform.rotozoom(pg.image.load(f"fig/beam.png"), angle, 2.0)
        self.vx = math.cos(math.radians(angle))
        self.vy = -math.sin(math.radians(angle))
        self.rect = self.image.get_rect()
        self.rect.centery = bird.rect.centery+bird.rect.height*self.vy
        self.rect.centerx = bird.rect.centerx+bird.rect.width*self.vx
        self.speed = 10

    def update(self):
        """
        ビームを速度ベクトルself.vx, self.vyに基づき移動させる
        引数 screen：画面Surface
        """
        self.rect.move_ip(self.speed*self.vx, self.speed*self.vy)
        if check_bound(self.rect) != (True, True):
            self.kill()


class Explosion(pg.sprite.Sprite):
    """
    爆発に関するクラス
    """
    def __init__(self, obj: "Bomb|Enemy", life: int):
        """
        爆弾が爆発するエフェクトを生成する
        引数1 obj：爆発するBombまたは敵機インスタンス
        引数2 life：爆発時間
        """
        super().__init__()
        img = pg.image.load(f"fig/explosion.gif")
        self.imgs = [img, pg.transform.flip(img, 1, 1)]
        self.image = self.imgs[0]
        self.rect = self.image.get_rect(center=obj.rect.center)
        self.life = life

    def update(self):
        """
        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
        爆発エフェクトを表現する
        """
        self.life -= 1
        self.image = self.imgs[self.life//10%2]
        if self.life < 0:
            self.kill()


class Enemy(pg.sprite.Sprite):
    """
    敵機に関するクラス
    """
    imgs = [pg.image.load(f"fig/alien{i}.png") for i in range(1, 4)]
```

```python

    def __init__(self):
        super().__init__()
        self.image = random.choice(__class__.imgs)
        self.rect = self.image.get_rect()
        self.rect.center = random.randint(0, WIDTH), 0
        self.vy = +6
        self.bound = random.randint(50, HEIGHT/2)  # 停止位置
        self.state = "down"  # 降下状態or停止状態
        self.interval = random.randint(50, 300)  # 爆弾投下インターバル

    def update(self):
        """
        敵機を速度ベクトルself.vyに基づき移動（降下）させる
        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
        引数 screen：画面Surface
        """
        if self.rect.centery > self.bound:
            self.vy = 0
            self.state = "stop"
        self.rect.centery += self.vy


class Score:
    """
    打ち落とした爆弾，敵機の数をスコアとして表示するクラス
    爆弾：1点
    敵機：10点
    """
    def __init__(self):
        self.font = pg.font.Font(None, 50)
        self.color = (0, 0, 255)
        self.value = 0
        self.image = self.font.render(f"Score: {self.value}", 0, self.color)
        self.rect = self.image.get_rect()
        self.rect.center = 100, HEIGHT-50

    def update(self, screen: pg.Surface):
        self.image = self.font.render(f"Score: {self.value}", 0, self.color)
        screen.blit(self.image, self.rect)


"""class Back_music:
    def __init__(self):
        pg.mixer.init()
        pg.mixer.music.load(f"fig/全てを創造する者「Dominus_Deus」.mp3")
    def music_play(self):
        pg.mixer.music.play(loops = 1)
        time.sleep(3)"""
```

```python
def main():
    pg.display.set_caption("真！こうかとん無双")
    screen = pg.display.set_mode((WIDTH, HEIGHT))
    bg_img = pg.image.load(f"fig/pg_bg.jpg")
    bg_img2 = bg_img
    bg_img3 = pg.image.load(f"fig/temple-3d1.jpg")
    bg_img4 = bg_img3
    score = Score()

    #Back_music(f"fig/全てを創造する者「Dominus_Deus」.mp3")

    bird = Bird(3, (900, 400))
    bombs = pg.sprite.Group()
```

ProjExD_Group02 / darkkokaton.py ↑ Top

Code Blame Raw

```python
def main():
    shield = pg.sprite.Group()
    tmr = 0
    clock = pg.time.Clock()
    while True:
        key_lst = pg.key.get_pressed()
        for event in pg.event.get():
            if event.type == pg.QUIT:
                return 0
            if event.type == pg.KEYDOWN and event.key == pg.K_SPACE:
                beams.add(Beam(bird))

        x = tmr%4800
        if score.value <= 50:  # スコアが50以下の時、
            pg.mixer.music.load(f"fig/全てを創造する者「Dominus_Deus」.mp3")  # BGMの音源をロード
            pg.mixer.music.play(-1)  # BGMを再生（無限ループ）
            pg.mixer.music.pause()  # このBGMはボスステージに流すので、ここでは一時停止
            screen.blit(bg_img, [-x, 0])  # 開幕背景
            screen.blit(bg_img2,[-x+1600, 0])
            screen.blit(bg_img, [-x+3200, 0])
            screen.blit(bg_img, [-x+4800, 0])
            tmr += 10
            clock.tick(200)
            if tmr%200 == 0:  # 200フレームに1回，敵機を出現させる
                emys.add(Enemy())
        else:  # スコアが50より大きいとき、ボスステージ
            pg.mixer.music.unpause()  # BGMの一時停止を解除
            screen.blit(bg_img3, [-x, 0])  # ボス戦背景
            screen.blit(bg_img4,[-x+1600, 0])
            screen.blit(bg_img3, [-x+3200, 0])
            screen.blit(bg_img3, [-x+4800, 0])
            clock.tick(100)
            tmr += 10
            clock.tick(200)


        #if tmr%200 == 0:  # 200フレームに1回，敵機を出現させる
            #emys.add(Enemy())

        for emy in emys:
            if emy.state == "stop" and tmr%emy.interval == 0:
```

```python
                # 敵機が停止状態に入ったら，intervalに応じて爆弾投下
                bombs.add(Bomb(emy, bird))

        for emy in pg.sprite.groupcollide(emys, beams, True, True).keys():
            exps.add(Explosion(emy, 100))  # 爆発エフェクト
            score.value += 10  # 10点アップ
            bird.change_img(6, screen)  # こうかとん喜びエフェクト

        for bomb in pg.sprite.groupcollide(bombs, beams, True, True).keys():
            exps.add(Explosion(bomb, 50))  # 爆発エフェクト

        for bomb in pg.sprite.groupcollide(bombs, shield, True, False).keys():
            exps.add(Explosion(bomb, 50))  # 爆発エフェクト
            score.value += 1  # 1点アップ

        for bomb in pg.sprite.spritecollide(bird, bombs, True):
            if bird.state == "hyper":
                exps.add(Explosion(bomb, 50))
                score.value += 1
            if bird.state == "normal":
                bird.change_img(8, screen) # こうかとん悲しみエフェクト
                score.update(screen)
                pg.display.update()
                time.sleep(2)
                return


        for event in pg.event.get():
            if (event.type == pg.KEYDOWN and event.key == pg.K_RSHIFT) and score.value >= 100:
                bird.state = "hyper"
                bird.hyper_life = 500
                score.value -= 100
                bird.update(key_lst, screen)

        for bomb in pg.sprite.spritecollide(bird, bombs, True):
            if bomb.state == "inactive":
                continue
            bird.change_img(8, screen) # こうかとん悲しみエフェクト
            score.update(screen)
            pg.display.update()
            time.sleep(2)
            return

        bird.update(key_lst, screen)
        beams.update()
        beams.draw(screen)
        emys.update()
        emys.draw(screen)
        bombs.update()
        bombs.draw(screen)
        exps.update()
        exps.draw(screen)
        gravity_fields.update()
        gravity_fields.draw(screen)
        score.update(screen)
        shield.update()
        shield.draw(screen)
        pg.display.update()
```

```
392             tmr += 1
393             clock.tick(50)
394
395
396
397     if __name__ == "__main__":
398         pg.init()
399         main()
400         pg.quit()
401         sys.exit()
402
```