ProjExD_Group05 / suta-_koukaton.py

c0a2303678 復活できるように設定 1f31be1 · 5 minutes ago

467 lines (387 loc) · 15.7 KB

Code Blame Raw

```python
#!/usr/bin/env python
import os
import random
from typing import List

# import basic pygame modules
import pygame as pg

# see if we can load more than standard BMP
if not pg.image.get_extended():
    raise SystemExit("Sorry, extended image module required")


# game constants
MAX_SHOTS = 1  # most player bullets onscreen
MAX_BOMBS = 1
ALIEN_ODDS = 22  # chances a new alien appears
BOMB_ODDS = 60  # chances a new bomb will drop
ALIEN_RELOAD = 12  # frames between new aliens
SCREENRECT = pg.Rect(0, 0, 640, 480)
SCORE = 0

main_dir = os.path.split(os.path.abspath(__file__))[0]


def load_image(file):
    """loads an image, prepares it for play"""
    file = os.path.join(main_dir, "data", file)
    try:
        surface = pg.image.load(file)
    except pg.error:
        raise SystemExit(f'Could not load image "{file}" {pg.get_error()}')
    return surface.convert()


def load_sound(file):
    """because pygame can be compiled without mixer."""
    if not pg.mixer:
        return None
    file = os.path.join(main_dir, "data", file)
    try:
        sound = pg.mixer.Sound(file)
```

```python
            return sound
        except pg.error:
            print(f"Warning, unable to load, {file}")
        return None


class Player(pg.sprite.Sprite):
    """
    Playerのイニシャライザ
    動作メソッド、
    銃の発射位置メソッドを生成しているクラス
    """

    speed = 5
    bounce = 24
    gun_offset = 0
    images: List[pg.Surface] = []

    def __init__(self, *groups):
        pg.sprite.Sprite.__init__(self, *groups)
        self.image = self.images[0]
        self.rect = self.image.get_rect(midbottom=SCREENRECT.midbottom)
        self.reloading = 0
        self.origtop = self.rect.top
        self.facing = -1

    def move(self, direction):
        if direction:
            self.facing = direction
        self.rect.move_ip(direction * self.speed, 0)
        self.rect = self.rect.clamp(SCREENRECT)
        if direction < 0:
            self.image = self.images[0]
        elif direction > 0:
            self.image = self.images[1]
        # self.rect.top = self.origtop - (self.rect.left // self.bounce % 2)

    def gunpos(self):
        pos = self.facing * self.gun_offset + self.rect.centerx
        return pos, self.rect.top


class Alien(pg.sprite.Sprite):
    """
    エイリアンのイニシャライザ
    動作メソッド
    銃の発射位置メソッド
    エイリアンの位置更新メソッドを生成しているクラス
    """

    speed = 5
    gun_offset = 0
    images: List[pg.Surface] = []

    def __init__(self, *groups):
        pg.sprite.Sprite.__init__(self, *groups)
        self.image = self.images[0]
        self.reloading = 0
```

```
            self.rect = self.image.get_rect(midtop=SCREENRECT.midtop)
            self.facing = -1
            self.origbottom = self.rect.bottom

        def move(self, direction):
            if direction:
                self.facing = direction
            self.rect.move_ip(direction * self.speed, 0)
            self.rect = self.rect.clamp(SCREENRECT)
            if direction < 0:
                self.image = self.images[0]
            elif direction > 0:
                self.image = self.images[1]

        def gunpos(self):
            pos = self.rect.centerx
            return pos, self.rect.bottom

        def update(self):
            #self.rect.move_ip(self.facing, 0)
            if not SCREENRECT.contains(self.rect):
                self.facing = -self.facing
                self.rect = self.rect.clamp(SCREENRECT)
            # self.frame = self.frame + 1
            # self.image = self.images[self.frame // self.animcycle % 3]


    class Explosion(pg.sprite.Sprite):
        """
        オブジェクトが衝突した際に爆発する演出を作成するクラス
        """

        defaultlife = 12
        animcycle = 3
        images: List[pg.Surface] = []

        def __init__(self, actor, *groups):
            pg.sprite.Sprite.__init__(self, *groups)
            self.image = self.images[0]
            self.rect = self.image.get_rect(center=actor.rect.center)
            self.life = self.defaultlife

        def update(self):
            """
            called every time around the game loop.

            Show the explosion surface for 'defaultlife'.
            Every game tick(update), we decrease the 'life'.

            Also we animate the explosion.
            """
            self.life = self.life - 1
            self.image = self.images[self.life // self.animcycle % 2]
            if self.life <= 0:
                self.kill()


    class Shot(pg.sprite.Sprite):
```

```
159         """
160         Playerが使う銃を生成するクラス
161         """
162 
163         speed = -10
164         images: List[pg.Surface] = []
165 
166         def __init__(self, pos, *groups):
167             pg.sprite.Sprite.__init__(self, *groups)
168             self.image = self.images[0]
169             self.rect = self.image.get_rect(midbottom=pos)
170 
171         def update(self):
172             """
173             called every time around the game loop.
174 
175             Every tick we move the shot upwards.
176             """
177             self.rect.move_ip(0, self.speed)
178             if self.rect.top <= 0:
179                 self.kill()
180 
181 
182     class Bomb(pg.sprite.Sprite):
183         """
184         Alienが落とす爆弾を生成するクラス
185         """
186 
187         speed = 10
188         images: List[pg.Surface] = []
189 
190         def __init__(self, alien_pos,*groups):
191             pg.sprite.Sprite.__init__(self, *groups)
192             self.image = self.images[0]
193             self.rect = self.image.get_rect(midtop=alien_pos)
194 
195         def update(self):
196             """
197             - make an explosion.
198             - remove the Bomb.
199             """
200             self.rect.move_ip(0, self.speed)
201             if self.rect.bottom >= SCREENRECT.bottom:
202                 self.kill()
203 
204 
205     class Score(pg.sprite.Sprite):
206         """
207         状況に応じて増減し、MAX_GUNSとMAX_BOMBSに関与するスコアクラス
208         """
209 
210         def __init__(self, *groups):
211             pg.sprite.Sprite.__init__(self, *groups)
212             self.font = pg.font.Font(None, 20)
213             self.font.set_italic(1)
214             self.color = "white"
215             self.lastscore = -1
216             self.update()
```

```python
            self.rect = self.image.get_rect().move(10, 450)

    def update(self):
        """We only update the score in update() when it has changed."""
        if SCORE != self.lastscore:
            self.lastscore = SCORE
            msg = f"Score: {SCORE}"
            self.image = self.font.render(msg, 0, self.color)


class Item(pg.sprite.Sprite):
    """
    ゲーム内でアイテムを表現するクラス。
    speed : int : アイテムの移動速度。
    images : List[pg.Surface] : アイテムを表現する画像のリスト。
    rect : pg.Rect : アイテムの位置とサイズを表す矩形。
    spawned : bool : アイテムが生成されたかどうかを示すフラグ。
    メソッド:
    update():アイテムの位置を更新し、画面端との衝突を処理する。
    spawn():アイテムを画面の中央に生成する。
    is_spawned() -> bool:アイテムが現在生成されているかどうかを確認する。
    collide_bombs(bombs: pg.sprite.Group) -> bool:爆弾との衝突を確認し、処理する。
    collide_shots(shots: pg.sprite.Group) -> bool:ショットとの衝突を確認し、処理する。
    reset():アイテムを初期状態にリセットする。
    """

    speed: int = 2 #itemの移動速度
    images: List[pg.Surface] = []#itemの画像リスト

    def __init__(self, *groups: pg.sprite.AbstractGroup) -> None:
        """
        Itemオブジェクトを初期化する。
        引数: *groups : pg.sprite.AbstractGroup : スプライトが所属するグループ。
        """
        pg.sprite.Sprite.__init__(self, *groups)
        self.image = pg.transform.scale(self.images[0], (64, 48))  # 画像サイズを変更
        self.image.set_colorkey((255, 255, 255))  # 背景を透明に設定
        self.rect = self.image.get_rect(center=SCREENRECT.center)  # 矩形を取得
        self.rect.topleft = (-100, -100)  # 初期位置を画面外に設定
        self.spawned = False  # アイテムが生成されたかどうかのフラグ

    def update(self) -> None:
        """
        アイテムの位置を更新し、画面端との衝突を処理する。
        """
        if self.spawned:
            self.rect.move_ip(self.speed, 0)  # アイテムを移動
            if self.rect.top > SCREENRECT.height:
                self.kill()  # 画面外に出たらアイテムを消す
                self.spawned = False  # フラグをリセット
            if self.rect.right >= SCREENRECT.right or self.rect.left <= 0:
                self.speed = -self.speed  # 画面端に当たったら移動方向を反転

    def spawn(self) -> None:
        """
        アイテムを画面の中央に生成する。
        """
        if not self.spawned:
```

```python
            self.rect.center = SCREENRECT.center  # アイテムを中央に移動
            self.spawned = True  # フラグを設定

    def is_spawned(self) -> bool:
        """
        アイテムが現在生成されているかどうかを確認する。
        戻り値: bool : アイテムが生成されていればTrue、そうでなければFalse。
        """
        return self.spawned

    def collide_bombs(self, bombs: pg.sprite.Group) -> bool:
        """
        爆弾との衝突を確認し、処理する。
        引数: bombs : pg.sprite.Group : 衝突を確認する爆弾のグループ。
        戻り値: bool : アイテムが爆弾と衝突した場合はTrue、そうでない場合はFalse。
        """
        if self.spawned:
            collided = pg.sprite.spritecollide(self, bombs, True)  # 衝突を確認

            if collided:
                self.kill()  # 衝突したらアイテムを消す
                self.spawned = False  # フラグをリセット
                self.rect.topleft = (-100, -100)  # 初期位置にリセット
                return True
        return False

    def collide_shots(self, shots: pg.sprite.Group) -> bool:
        """
        ショットとの衝突を確認し、処理する。
        引数: shots : pg.sprite.Group : 衝突を確認するショットのグループ。
        戻り値: bool : アイテムがショットと衝突した場合はTrue、そうでない場合はFalse。
        """
        if self.spawned:
            collided = pg.sprite.spritecollide(self, shots, True)
            if collided:
                self.kill()
                self.spawned = False  # 衝突したらフラグをリセット
                self.rect.topleft = (-100, -100)  # 画面外の初期位置にリセット
                return True
        return False

    def reset(self) -> None:
        """
        アイテムを初期状態にリセットする。
        """
        self.spawned = False # フラグをリセット
        self.rect.topleft = (-100, -100)  # 画面外に初期位置をリセット


def main(winstyle=0):
    # Initialize pygame
    if pg.get_sdl_version()[0] == 2:
        pg.mixer.pre_init(44100, 32, 2, 1024)
    pg.init()
    if pg.mixer and not pg.mixer.get_init():
        print("Warning, no sound")
        pg.mixer = None

```

```
333         fullscreen = False
334         winstyle = 0  # |FULLSCREEN
335         bestdepth = pg.display.mode_ok(SCREENRECT.size, winstyle, 32)
336         screen = pg.display.set_mode(SCREENRECT.size, winstyle, bestdepth)
337 
338         # Load images, assign to sprite classes
339         img = load_image("3.png")
340         Player.images = [img, pg.transform.flip(img, 1, 0)]
341         img = load_image("explosion1.gif")
342         Explosion.images = [img, pg.transform.flip(img, 1, 1)]
343         Alien.images = [load_image(im) for im in ("alien1.gif", "alien2.gif", "alien3.gif")]
344         Bomb.images = [load_image("bomb.gif")]
345         Shot.images = [load_image("shot.gif")]
346         Item.images = [load_image("item.png")]  # アイテム画像を読み込む
347 
348         icon = pg.transform.scale(Alien.images[0], (32, 32))
349         pg.display.set_icon(icon)
350         pg.display.set_caption("Pygame Aliens")
351         pg.mouse.set_visible(0)
352 
353         bgdtile = load_image("utyuu.jpg")
354         background = pg.Surface(SCREENRECT.size)
355         background.blit(bgdtile, (0, 0))
356         screen.blit(background, (0, 0))
357         pg.display.flip()
358 
359         boom_sound = load_sound("boom.wav")
360         shoot_sound = load_sound("car_door.wav")
361         if pg.mixer:
362             music = os.path.join(main_dir, "data", "house_lo.wav")
363             pg.mixer.music.load(music)
364             pg.mixer.music.play(-1)
365 
366         player = pg.sprite.Group()
367         aliens = pg.sprite.Group()
368         shots = pg.sprite.Group()
369         bombs = pg.sprite.Group()
370         items = pg.sprite.Group()
371         all = pg.sprite.RenderUpdates()
372 
373         global SCORE
374         player = Player(all)
375         alien = Alien(aliens, all)
376         item = Item(items, all)  # アイテムを初期化し追加
377 
378         if pg.font:
379             all.add(Score(all))
380 
381         item_spawn_time = random.randint(300, 600)  # 初回のアイテム出現時間をランダムに設定（5秒から10秒）
382         item_timer = 0
383         item_spawned = False
384 
385         clock = pg.time.Clock()
386 
387         while player.alive() and alien.alive():
388             for event in pg.event.get():
389                 if event.type == pg.QUIT:
390                     return
```

```
391                 if event.type == pg.KEYDOWN and event.key == pg.K_ESCAPE:
392                     return
393                 if event.type == pg.KEYDOWN:
394                     if event.key == pg.K_f:
395                         if not fullscreen:
396                             print("Changing to FULLSCREEN")
397                             screen_backup = screen.copy()
398                             screen = pg.display.set_mode(SCREENRECT.size, winstyle | pg.FULLSCREEN, best
399                             screen.blit(screen_backup, (0, 0))
400                         else:
401                             print("Changing to windowed mode")
402                             screen_backup = screen.copy()
403                             screen = pg.display.set_mode(SCREENRECT.size, winstyle, bestdepth)
404                             screen.blit(screen_backup, (0, 0))
405                         pg.display.flip()
406                         fullscreen = not fullscreen
407 
408             keystate = pg.key.get_pressed()
409 
410             all.clear(screen, background)
411             all.update()
412 
413             direction = keystate[pg.K_RIGHT] - keystate[pg.K_LEFT]
414             player.move(direction)
415             firing = keystate[pg.K_SPACE]
416             if not player.reloading and firing and len(shots) < MAX_SHOTS:
417                 Shot(player.gunpos(), shots, all)
418                 if pg.mixer and shoot_sound is not None:
419                     shoot_sound.play()
420             player.reloading = firing
421 
422             direction = keystate[pg.K_d] - keystate[pg.K_a]
423             alien.move(direction)
424             firing = keystate[pg.K_t]
425             if not alien.reloading and firing and len(bombs) < MAX_BOMBS:
426                 Bomb(alien.gunpos(), bombs, all)
427                 if pg.mixer and shoot_sound is not None:
428                     shoot_sound.play()
429             alien.reloading = firing
430 
431             for shot in pg.sprite.spritecollide(alien, shots, 1):
432                 Explosion(shot, all)
433                 Explosion(alien, all)
434                 if pg.mixer and boom_sound is not None:
435                     boom_sound.play()
436                 alien.kill()
437 
438             for bomb in pg.sprite.spritecollide(player, bombs, 1):
439                 Explosion(bomb, all)
440                 Explosion(player, all)
441                 if pg.mixer and boom_sound is not None:
442                     boom_sound.play()
443                 player.kill()
444 
445             item_timer += 1# アイテム生成タイマーを更新
446             if not item_spawned and item_timer >= item_spawn_time:
447                 item.spawn()  # アイテムを生成
448                 item_spawned = True
```

```
449
450             # アイテムが爆弾と衝突したかを確認
451             if item.collide_bombs(bombs) or item.collide_shots(shots):
452                 item_timer = 0
453                 item_spawn_time = random.randint(300, 600)  # 新しいアイテム出現時間を設定
454                 item = Item(items, all)  # アイテムを初期化し再度作成
455                 item_spawned = False
456
457             pg.display.update(all.draw(screen))
458
459             clock.tick(40)
460
461         if pg.mixer:
462             pg.mixer.music.fadeout(1000)
463         pg.time.wait(1000)
464
465     if __name__ == "__main__":
466         main()
467         pg.quit()
```