Wave-Born

Pedro Iglésias 89318

Wei Ye 93442

Funcionamento do jogo

- Pode-se controlar o player atrávez das teclas 'a', 'd', e 'space', as primeiras duas movem o player para esquerda ou direita enquanto a ultima é para saltar.
- O mundo está totalmente escuro, não se consegue ver nada, nem paredes nem monstros. Quando um monstro começar a atacar o player ou simplemente gritar, uma onda que revela um pedaço do mundo é gerada. Além disso, o movimento do player também gera ondas.
- Qualquer monstro pode matar o player e vice-versa.

Arquitetura

- chunks: contém ficheiros json para geração do mundo.
- menu: contém classes de menu principal, configuração e pause.
- models: contém objetos usados no jogo como player, monster e wave, etc.
- sources: contém imagens e sons que o jogo precisa.
- sprites: contém classes de sprite.

Command

Command foi usado para interpertar os comandos do utilizador a partir das teclas configuradas a partir do menu.

```
def controls(self, left, right, jump):
    self.control_keys = {left: Left, right: Right, jump: Up}
    self.control_keys_name = {'left': left, 'right': right, 'jump': jump}

def command(self, control):
    if control in self.control_keys.keys():
        cmd = self.control_keys[control]()
        cmd.execute(self)
        return cmd

def move(self, direction: Directions = None):
    """Add one piece, pop one out."""
    if direction:
        self.direction = direction

@property
def x(self):
    return self.pos[0]

@x.setter
def x(self, v):
    self.pos = v, self.y

@property
def y(self):
    return self.pos[1]

@y.setter
def y(self, v):
    self.pos = self.x, v

@property
def left_key(self):
    return self.control_keys_name['left']

@property
def right_key(self):
    return self.control_keys_name['right']
```

Flyweight

Flyweight foi usado em quase todos sprites tanto na geração do mundo como nos monstros, para reutilizar as texturas em vez estar sempre a cria-las.

```
class BlockSprite(pygame.sprite.Sprite):
    def __init__(self, chunk, blocks_x, blocks_y, SCALE):
        Sprite.__init__(self)
        BLOCK_SPRITESHEET = SpriteSheet("sources/imgs/blocks.png")
        self.chunk = chunk
        self.blocks = self.chunk.blocks

        images = [pygame.transform.scale(BLOCK_SPRITESHEET.image_at((SCAL
                                         (SCALE, SCALE), ) for block in s

        rects = [image.get_rect() for image in images]
        for idx, rect in enumerate(rects):
            rect.x = self.blocks[idx].x
            rect.y = self.blocks[idx].y

        self.rect = rects[0].copy()
        for rect in rects[1:]:
            self.rect.union_ip(rect)

        # Create a new transparent image with the combined size.
        self.image = pygame.Surface(self.rect.size, pygame.SRCALPHA)
        # Now blit all sprites onto the new surface.
        for idx, image in enumerate(images):
            self.image.blit(image, (rects[idx].x - self.rect.left,
                                    rects[idx].y - self.rect.top))
        self.mask = pygame.mask.from_surface(self.image)

    def move(self, velocity):
        self.rect.move_ip(velocity)
        try:
            self.chunk.end_sprite.move(velocity)
        except:
            pass

    def remove(self):
        self.kill()
```

Prototype

Prototype foi usado para monstros e diferenciar entre monstros voadores e de chão.

Como também um spawner para fazer spawn aos monstros.

```
class BirdLike(Monster):
    def __init__(self, w
                 stop_he
                 jump_di
        super().__init__
```

```
class Monster:
    _ID = 0
```

```
class Whale(Monster):
    def __init__(self, wid
                 stop_heig
```

```
class GroundMonster(Monster):
    TRANSITIONS = {
        Event.ATTACK: [Transiti
        Event.JUMP: [Transition
        Event.FAIL: [Transition
```

```
class TurtleLike(GroundMonster):
    def __init__(self, width, hei
```

```
class SpiderLike(GroundMonster):
    def __init__(self, width, heig
```

```
class Spawner:
    def spawn_monster(self, prototype) -> Monster:
        return prototype.clone()
```

Singleton

Todos os sprites são singletons pois não faz sentido ser criado varios objetos sprites quando a textura é sempre a mesma que esta a ser usada e a única diferença é que temos um novo monstro modelo a ser adicionado ou removido desta classe.

O world é um singleton pois o mundo é sempre o mesmo e os chunks que são usados para gerar o mundo são sempre os mesmos a única diferença é a ordem dos chunks gerados.

```
class MonsterSprite(pygame.sprite.Sprite):
    _singleton = None

    def __init__(self, image_update_per_frames=0, pos_update_per_frames=0):
        Sprite.__init__(self)
        self.monsters: list
        self.rects: dict
        self.left_move_images: list
        self.right_move_images: list
        self.img_indexes: dict
        self.left_dead_images: list
        self.right_dead_images: list
        self.left_attack_images: list
        self.right_attack_images: list
        self.image_update_per_frames = image_update_per_frames
        self.pos_update_per_frames = pos_update_per_frames
        self.image_update_count = 0
        self.pos_update_count = 0
```

```
class World:
    _singleton = None

    def __init__(self, difficulty, time_limit,blocks_x,blocks_y,SCALE):
        self.blocks_x = blocks_x
        self.blocks_y = blocks_y
        self.SCALE = SCALE

        self.loaded_chunks = self.loadFiles()

        self.generateWorld(self.loaded_chunks,difficulty, time_limit)
        World._singleton = self

    def loadFiles(self):
        self.start = Chunk.load_chunk(0, "./chunks/normal/plane")
        self.end = Chunk.load_chunk(0, "./chunks/normal/plane", end=True

        file_path = "./chunks/normal"
        normal_chunks = [Chunk.load_chunk(0, join(file_path, f)) for f in
                         isfile(join(file_path, f))]

        file_path = "./chunks/tunnel"
        tunnel_chunks = [Chunk.load_chunk(0, join(file_path, f))
                         isfile(join(file_path, f))]
        return (normal_chunks, tunnel_chunks)
```

State

State foram usados para saber todas transições e estados em que o player e os monstros estão.

```python
class State:
    def __init__(self, name) -> None:
        self.name = name

    @classmethod
    def enter(cls, object):
        logging.debug(f"{object} Entering {cls.__name__}")

    @classmethod
    def update(cls, object):
        pass

    @classmethod
    def exit(cls, object):
        pass


class Transition:
    def __init__(self, _from, _to) -> None:
        self._from = _from
        self._to = _to


class FSM:
    def __init__(self, states: list[State], transitions: dict[State, list[Transi
        self._states = states
        self._transitions = transitions

        self.current: State = self._states[0]
        self.end: State = self._states[-1]
```

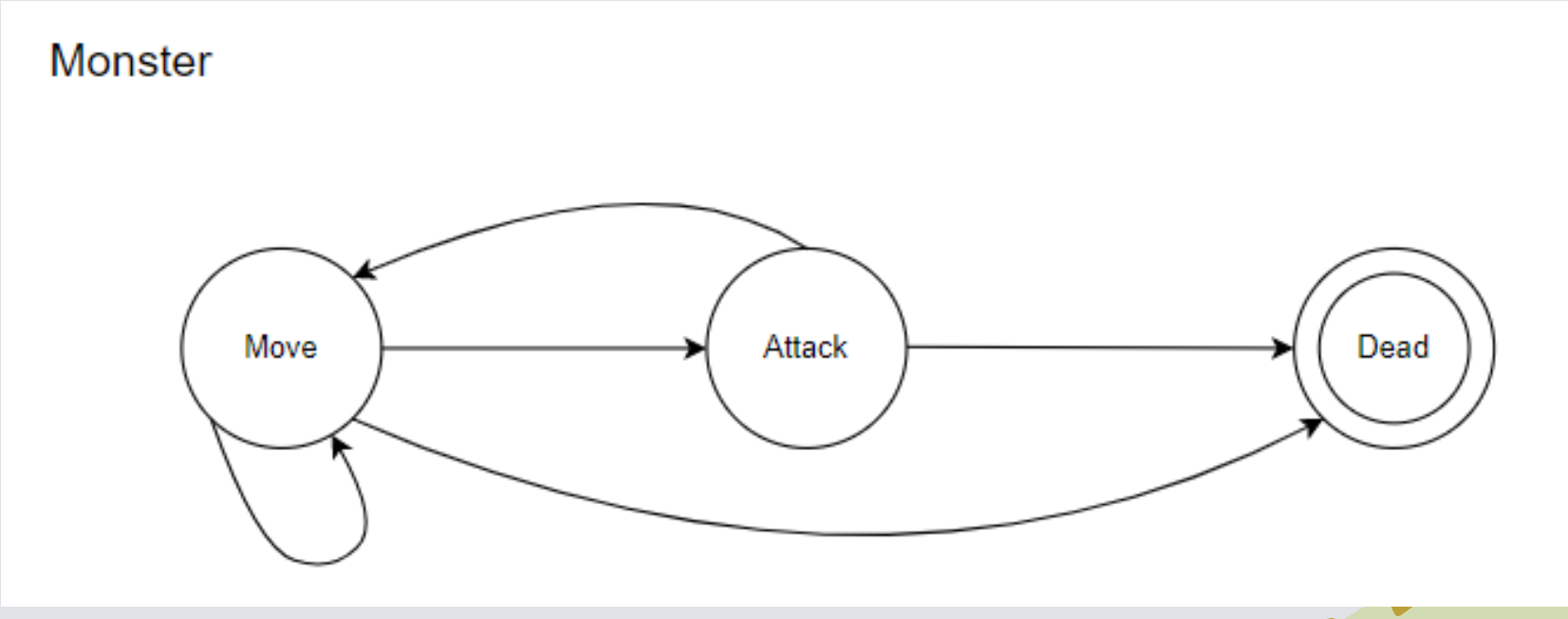
States de Monster

Um monstro em geral tem 3 estados: Move, Attack e Dead.

No estado Move, o monstro move-se horizontalmente.

Existe uma probabilidade x de passar do estado Move para Attack.

O montro pode morrer em qualquer estado.



FSM

FSM foi usado para controlar transições entre estados.

```
def update(self, **kwargs):
    super(GroundMonster, self).update(**kwargs)
    event = None
    player = Player.SPRITE
    if player.stepped_on(self.sprite.rect):
        event = Event.DYING
    elif self.out_of_world():
        event = Event.DEAD
    elif self.fsm.current == Jump and self.falling:
        event = Event.FALL
    elif self.fsm.current == Fall and not self.falling:
        event = Event.MOVE
        self.fail_speed = 1
    elif self.fsm.current == Attack:
        event = Event.JUMP
    elif self.fsm.current == Move and not self.step_on_wall():
        event = Event.MOVE_IN_AIR
    elif self.fsm.current == Move and \
            abs(player.y - self.y) < 32 and abs(player.x - self.x) < 128 and self.want_attack():
        if not self.attacking:
            if player.rect.x < self.x:
                self.direction = -1
            else:
                self.direction = 1
            event = Event.ATTACK
    elif self.fsm.current == MoveInAir:
        event = Event.FALL
        self.fail_speed = 2

    self.fsm.update(event, self)
```

States de GroundMonster

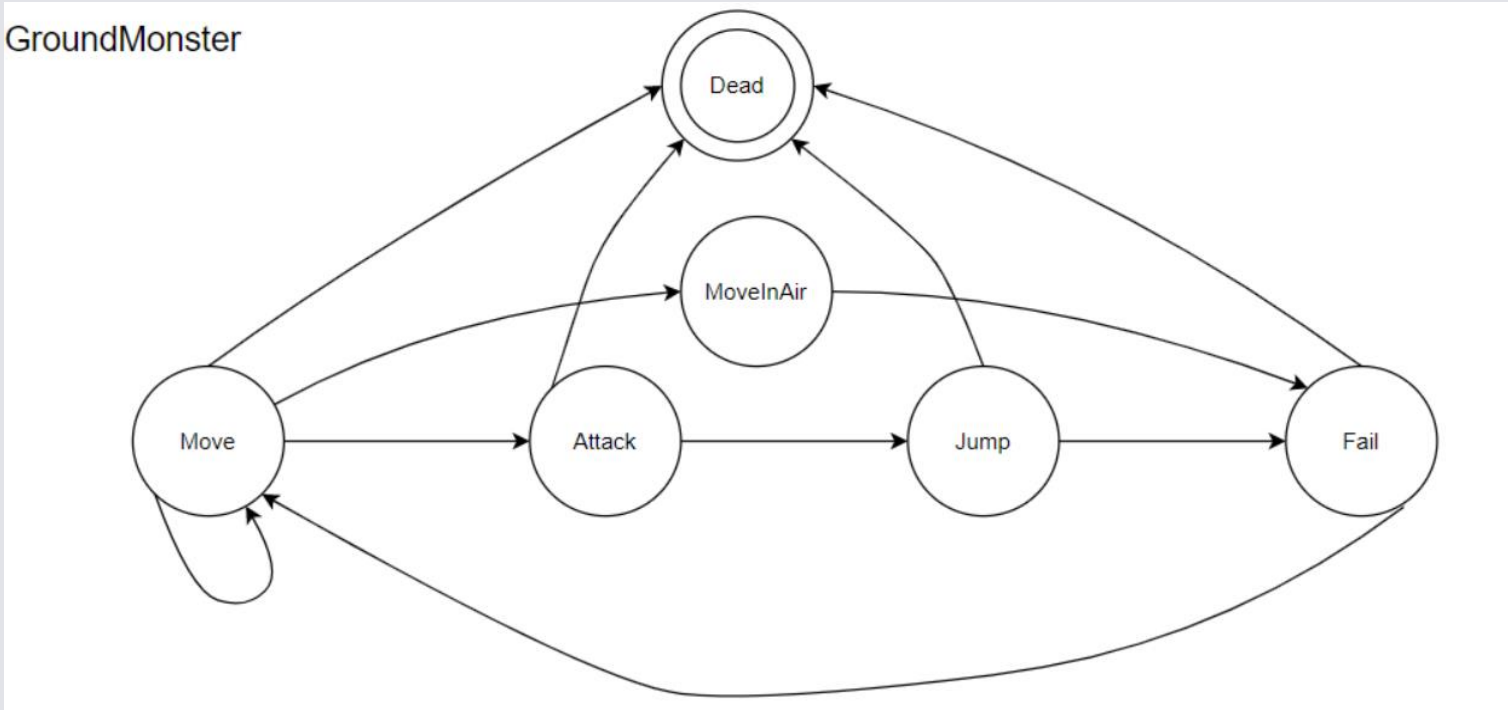
Um monstro em geral tem 6 estados: Move, Attack, Jump, Fall, MoveInAir e Dead.

No estado Move, o monstro move-se horizontalmente.

Existe uma probabilidade x de passar do estado Move para Attack.

O monstro pode morrer em qualquer estado.

MoveInAir acontece quando o monstro sai do bloco, ou seja, está a caminhar no ar.



Game loop

Game loop foi usado para cada criação de um mundo.

Passando por "process input": onde os eventos e o input do teclado do player é processado.

Depois por "update game": onde tudo no jogo é updated os monstros, o player, o mundo e a camera.

Por fim passa por "render": onde os sprites todos são renderizados.

```
#process input
for e in event.get():
    if e.type == QUIT:
        pygame.quit()
        break

    elif e.type == KEYDOWN or e.type == KEYUP:
        if e.key == K_ESCAPE and not opened_menu:
            menu.set_show()
            opened_menu = True
        else:
            opened_menu = False

        lastKey = pygame.key.get_pressed()
        # player.command(e.key)

if menu.exit:
    pygame.quit()
    break

player.controls(menu.left_key, menu.right_key, menu.jump_key)

if lastKey:
    if lastKey[player.left_key]:
        player.command(player.left_key)
        if (world.current_chunk <= 0 or world.current_chunk >=
            camera_move = False
        movement = -1
    if lastKey[player.right_key]:
        player.command(player.right_key)
        if (world.current_chunk >= len(world.world_chunks) - 1
            camera_move = False
        movement = 1
```

```
#render
all_sprites.update()
all_sprites.draw(screen)
bird_sprite.update()
bird_sprite.draw(screen)
spider_sprite.update()
spider_sprite.draw(screen)
turtle_sprite.update()
turtle_sprite.draw(screen)
whale_sprite.update()
whale_sprite.draw(screen)

for wave in waves:
    wave.draw(mask)

if not hardmode:
    player_sprite.draw(mask)

# draw transparent circle and update display
screen.blit(mask, (0, 0))
```

```
#update game

if(random.randint(0,100) < 5 and 3 > (len(bird_sprite.monsters)
    selectMonster = random.randint(0,100)
    if(selectMonster <= 19):
        whale_sprite._add_monster(spawner.spawn_monster(whale))
    elif(selectMonster <= 44):
        bird_sprite._add_monster(spawner.spawn_monster(bird))
    elif(selectMonster <= 69):
        spider_sprite._add_monster(spawner.spawn_monster(spider
    elif(selectMonster <= 100):
        turtle_sprite._add_monster(spawner.spawn_monster(turtle

# create cover surface
mask.fill(0)

# world interaction
moved += movement
if int(moved / (SCALE * 16)) == 1:
    removed, added = world.loadNextChunk()
    if removed:
        all_sprites.remove(removed)
    if added:
        all_sprites.add(added)
        try:
            all_sprites.add(added.chunk.end_sprite)
        except:
            pass
    moved = 0
elif (int(moved / (SCALE * 16)) == -1):
    removed, added = world.loadPrevChunk()
    if removed:
```

Aspectos "inovadores"

Os aspetos mais interessantes do jogo são as ondas de som que revelam parte do mapa e a geração do mundo.

O som revelar o mapa de um jogo é usado em diversos jogos mas nunca tinhamos visto num platformer por isso é que decidimos aplicar neste jogo.

A geração do mundo é feito antes de começar um new game e utiliza ficheiros existentes de chunks pré-fabricados e monta um mapa possivel de completar do inicio ao fim a partir de condições no ficheiro pré-determinadas.

Depois o mundo como já tem os chunks escolhidos e gerados só precisa de carregar os 3 chunks a volta do player (o chunk em que o player esta 2 para tras e 2 para a frente) funcionando como uma conveyor belt onde o player caminha numa direção e vai carregando novos chunks à medida que caminha.

GitHub Link

https://github.com/Iglesias-Leafwind/Wave-Born/tree/master

Referencias

- Inspiração da mecânica do som Dark Echo:
  https://www.youtube.com/watch?v=tuOC8oTrFbM
- Sprites:
  - Background.png -> https://wallpapersden.com/cyberpunk-city-pixel-art-wallpaper/1360x768/
  - Bird.png -> https://www.nicepng.com/ourpic/u2q8q8i1q8w7r5i1_sprite-sheet-bird-png/
  - End_city.png -> https://www.deviantart.com/mysticmorning/art/Sci-Fi-Fantasy-Building-2-359889492
  - Feather.png -> http://clipart-library.com/clip-art/transparent-feather-21.htm
  - Player.png -> https://www.pixilart.com/art/2d-player-sprite-sheet-317ef5787732657
  - Spider.png -> https://www.pngitem.com/middle/hixJxbT_spider-0-spider-sprite-animation-sheet-hd-png/
  - Tortoise.png -> https://www.spriters-resource.com/ds_dsi/finalfantasy12revenantwings/sheet/424/
  - Whale.png -> https://opengameart.org/content/swimming-whale
- Sons
  - Bird.mp3 -> https://pixabay.com/sound-effects/gryffin-cry-6995/
  - Breeze_bay.mp3 -> https://soundcloud.com/hellometeor/breeze-bay
  - Jump.mp3 -> https://pixabay.com/sound-effects/swing-whoosh-110410/
  - Land.mp3 -> https://pixabay.com/sound-effects/land2-43790/
  - Running.mp3 -> https://pixabay.com/sound-effects/running-1-6846/
  - Step.mp3 -> https://pixabay.com/sound-effects/footsteps-grass-1-6810/
  - Turtle.mp3 -> https://pixabay.com/sound-effects/sleeping-monster-38084/
  - Whale.mp3 -> https://pixabay.com/sound-effects/long-howl-whale-and-monster-37270/