Copyleaks

Plagiarism report

entities.py

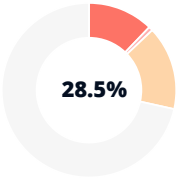
Scan details

Scan time:
April 13th, 2024 at 9:29 UTC

Total Pages:
17

Total Words:
4132

Plagiarism Detection



Types of plagiarism		Words
Identical	12.1%	502
Minor Changes	0.8%	32
Paraphrased	15.6%	643
Omitted Words	0%	0

AI Content Detection

Text coverage
- AI text
- Human text

Plagiarism Results: (9)

Pygame Platformer 05 — Animation & Level Design I | by Yen | Medium — 19%

https://medium.com/@monlesyen/pygame-platformer-05-animation-level-design-i-35a00901a64

Yen

Open in appSign up Sign in Write Sign up Sign in Pygame Platformer 05 — Animation & Level Design I Y...

python - Pygame Platformer Collisions - Stack Overflow — 17.7%

https://stackoverflow.com/questions/78196410/pygame-platformer-collisions

Stack Overflow About ...

Pygame Platformer 03 — Physics, Output data of rect, Jump | by Yen | M... — 12.4%

https://medium.com/@monlesyen/pygame-platformer-03-physics-output-data-of-rect-jump-d75947dae66

Yen

Open in appSign up Sign in Write Sign up Sign in Pygame Platformer 03 — Physics, Output data of rect, Jump ...

python - Projectile motion and gravity in pygame only sometimes workin... — 5.4%

https://stackoverflow.com/questions/55822116/projectile-motion-and-gravity-in-pygame-only-sometimes-wor...

Stack Overflow About ...

Results

python - Tried running a program and got this error - Stack Overflow — 1.8%

https://stackoverflow.com/questions/62649576/tried-running-a-program-and-got-this-error

Stack Overflow About ...

Pyhton Kivy Tutorial 6 - Customize and Re-Use Widgets — 0.8%

https://kraisnet.de/index.php/en/?view=article&id=72&catid=20

Haiko's Raspberry Blog ...

---

python - How to make the border appear when hovering over a button in... — 0.8%

https://stackoverflow.com/questions/77511803/how-to-make-the-border-appear-when-hovering-over-a-butto...

Stack Overflow About ...

---

python - How to show button02 .kv-defined appearance upon button 01 ... — 0.8%

https://stackoverflow.com/questions/76422266/how-to-show-button02-kv-defined-appearance-upon-button-...

Stack Overflow About ...

Example for MutatorMath in Drawbot. It shows how to make math objec... — 0.8%

https://gist.github.com/letterror/6565405279547124e992

Skip to content ...

```python
import math
import random
import pygame

from scripts.particle import Particle

class PhysicsEntity:
    """
    A base class for entities in the game that interact physically with the world, including players, enemies, a

    Attributes:
        game: Reference to the main game object for accessing global resources and settings.
        type (str): The type of the entity, used to distinguish between different kinds of entities.
        pos (list): The position of the entity in the game world as [x, y].
        size (tuple): The size of the entity as (width, height).
        velocity (list): The current velocity of the entity as [vx, vy].
        collisions (dict): A dictionary tracking collision states in four directions: 'up', 'down', 'left', 'rig
        action (str): The current action or state of the entity, affecting its behavior and appearance.
        anim_offset (tuple): Offset for the entity's animation relative to its position, as (offset_x, offset_y)
        flip (bool): Indicates if the entity's sprite should be flipped horizontally.
        last_movement (list): The last movement input received, as [dx, dy], influencing the entity's direction.
    """

    def __init__(self, game, e_type, pos, size):
        """
        Initializes a new PhysicsEntity with given properties and default values for velocity, collisions, and a

        Parameters:
            game: The main game object.
            e_type (str): The entity type.
            pos (list): The initial position of the entity.
            size (tuple): The size of the entity.
        """
        # Inicializace fyzikální entity se všemi jejími vlastnostmi
        self.game = game
        self.type = e_type
        self.pos = list(pos)  # Pozice entity ve hře
        self.size = size  # Velikost entity
        self.velocity = [0, 0]  # Rychlost pohybu entity
        self.collisions = {'up': False, 'down': False, 'right': False, 'left': False}  # Informace o kolizích s
        self.action = ''  # Aktuální akce entity
        self.anim_offset = (-3, -3)  # Odsazení animace od pozice entity
        self.flip = False  # Určuje, zda je entity otočená
        self.set_action('idle')  # Nastavení výchozí akce na "klid"
        self.last_movement = [0, 0]  # Poslední směr pohybu entity

    def rect(self):
        """
        Creates a pygame.Rect object representing the entity's current position and size.

        Returns:
            pygame.Rect: The rectangle representing the entity's bounds.
        """
        # Vrátí obdélníkový objekt reprezentující pozici a velikost entity
        return pygame.Rect(self.pos[0], self.pos[1], self.size[0], self.size[1])

    def set_action(self, action):
        """
        Sets the current action of the entity, updating its animation accordingly.

        Parameters:
            action (str): The new action to set for the entity.
        """
        # Nastaví akci entity na zadanou
        if action != self.action:
            self.action = action
            self.animation = self.game.assets[self.type + '/' + self.action].copy()  # Zkopírování animace pro o

    def update(self, tilemap, movement=(0, 0)):
        """
        Updates the entity's state, including position, velocity, collisions, and animation based on movement in

        Parameters:
            tilemap: The tilemap object the entity interacts with.
            movement (tuple): The movement input for the entity as (dx, dy).
        """
        # Aktualizuje stav entity na základě pohybu a kolizí s okolím
        self.collisions = {'up': False, 'down': False, 'right': False, 'left': False}  # Resetování informací o
```

```
        # Pohyb entity v horizontálním směru
        frame_movement = (movement[0] + self.velocity[0], movement[1] + self.velocity[1])
        self.pos[0] += frame_movement[0]
        entity_rect = self.rect()

        # Detekce kolizí s dlaždicemi v horizontálním směru
        for rect in tilemap.physics_rects_around(self.pos):
            if entity_rect.colliderect(rect):
                if frame_movement[0] > 0:
                    entity_rect.right = rect.left
                    self.collisions['right'] = True
                if frame_movement[0] < 0:
                    entity_rect.left = rect.right
                    self.collisions['left'] = True
                self.pos[0] = entity_rect.x

        # Pohyb entity ve vertikálním směru
        self.pos[1] += frame_movement[1]
        entity_rect = self.rect()

        # Detekce kolizí s dlaždicemi ve vertikálním směru
        for rect in tilemap.physics_rects_around(self.pos):
            if entity_rect.colliderect(rect):
                if frame_movement[1] > 0:
                    entity_rect.bottom = rect.top
                    self.collisions['down'] = True
                if frame_movement[1] < 0:
                    entity_rect.top = rect.bottom
                    self.collisions['up'] = True
                self.pos[1] = entity_rect.y

        # Otočení entity v případě změny směru pohybu
        if movement[0] > 0:
            self.flip = False
        if movement[0] < 0:
            self.flip = True

        self.last_movement = movement

        # Aktualizace rychlosti entity ve vertikálním směru pod vlivem gravitace
        self.velocity[1] = min(5, self.velocity[1] + 0.1)

        # Zastavení entity v případě kolize s podložkou
        if self.collisions['down'] or self.collisions['up']:
            self.velocity[1] = 0

        # Aktualizace animace entity
        self.animation.update()

    def render(self, surf, offset=(0, 0)):
        """
        Renders the entity on the specified surface, adjusted by the given offset and flipped based on the entit

        Parameters:
            surf (pygame.Surface): The surface to draw the entity on.
            offset (tuple): The offset to apply to the entity's position when drawing.
        """
        # Vykreslení animace entity na herní plochu s ohledem na pozici pozorovatele (offset)
        surf.blit(pygame.transform.flip(self.animation.img(), self.flip, False), (self.pos[0] - offset[0] + self

class Enemy(PhysicsEntity):
    """
    Represents an enemy entity with specific behaviors like patrolling or attacking the player.

    Inherits from PhysicsEntity and adds enemy-specific attributes and methods.

    Attributes:
        walking (int): Counter for controlling patrolling behavior or other movement patterns.
    """

    def __init__(self, game, pos, size):
        """
        Initializes a new Enemy instance with the specified properties.

        Parameters:
            game: The main game object.
            pos (list): The initial position of the enemy.
```

```
            size (tuple): The size of the enemy.
        """
        # Inicializace nepřátelské entity s předkem PhysicsEntity
        super().__init__(game, 'enemy', pos, size)

        # Počet kroků, které má nepřítel udělat ve směru
        self.walking = 0

    def update(self, tilemap, movement=(0, 0)):
        """
        Updates the enemy's state, including patrolling behavior, attacks, and interactions with the player.

        Parameters:
            tilemap: The tilemap object the enemy interacts with.
            movement (tuple): The movement input for the enemy as (dx, dy).
        """
        # Aktualizace stavu nepřátelské entity na základě pohybu a kolizí s okolím
        if self.walking:
            # Pokud má nepřítel provádět kroky, provede se pohyb v daném směru
            if tilemap.solid_check((self.rect().centerx + (-7 if self.flip else 7), self.pos[1] + 23)):
                # Pokud narazí na překážku, zkontroluje, zda může změnit směr
                if (self.collisions['right'] or self.collisions['left']):
                    self.flip = not self.flip
                else:
                    movement = (movement[0] - 0.5 if self.flip else 0.5, movement[1])
            else:
                self.flip = not self.flip
            self.walking = max(0, self.walking - 1)
            if not self.walking:
                # Po dokončení kroků se provede kontrola hráče a případné vystřelení
                dis = (self.game.player.pos[0] - self.pos[0], self.game.player.pos[1] - self.pos[1])
                if (abs(dis[1]) < 16):
                    if (self.flip and dis[0] < 0):
                        self.game.sfx['shoot'].play()

                        self.game.fireball.append([[self.rect().centerx - 7, self.rect().centery], -1.5, 0])

                    if (not self.flip and dis[0] > 0):
                        self.game.sfx['shoot'].play()

                        self.game.fireball.append([[self.rect().centerx + 7, self.rect().centery], 1.5, 0])

        elif random.random() < 0.01:
            # Náhodné začátky kroků, pokud nepřítel nestojí
            self.walking = random.randint(30, 120)

        # Aktualizace pohybu entity
        super().update(tilemap, movement=movement)

        # Nastavení akce podle směru pohybu
        if movement[0] != 0:
            self.set_action('run')
        elif movement[0] == 0:
            self.set_action('idle')


        # Detekce úderu od hráče a reakce
        if abs(self.game.player.dashing) >= 20:

            if self.rect().colliderect(self.game.player.rect()):
                self.game.sfx['hit'].play()
                for i in range(20):
                    angle = random.random() * 3.14 * 2
                    speed = random.random() * 5
                    self.game.particles.append(Particle(self.game, 'particle', self.rect().center, velocity=[mat
                return True

    def render(self, surf, offset=(0, 0)):
        """
        Renders the enemy entity on the specified surface, adjusted by the given offset and with additional enem

        Parameters:
            surf (pygame.Surface): The surface to draw the enemy on.
            offset (tuple): The offset to apply to the enemy's position when drawing.
        """
        # Vykreslení nepřátelské entity s přihlédnutím k otočení
        super().render(surf, offset=offset)
```

```
        if self.flip:
            # Vykreslení zbraně v případě otočení entity
            surf.blit(pygame.transform.flip(self.game.assets['weapon1'], True, False), (self.rect().centerx - 6
        else:
            surf.blit(self.game.assets['weapon1'], (self.rect().centerx + 4 - offset[0], self.rect().centery - o

class Player(PhysicsEntity):
    """
    Represents the player entity with specific behaviors like jumping, dashing, and interaction with game elemen

    Inherits from PhysicsEntity and adds player-specific attributes and methods.

    Attributes:
        air_time (int): Counter for tracking how long the player has been in the air (for jump control).
        jumps (int): The number of available jumps (for double jump or similar mechanics).
        wall_slide (bool): Indicates whether the player is currently sliding down a wall.
        dashing (int): Counter for dash action, controlling its behavior and duration.
    """

    def __init__(self, game, pos, size):
        """
        Initializes a new Player instance with the specified properties.

        Parameters:
            game: The main game object.
            pos (list): The initial position of the player.
            size (tuple): The size of the player.
        """
        # Inicializace hráčovy entity s předkem PhysicsEntity
        super().__init__(game, 'player', pos, size)

        # Počet snímků, které hráč stráví ve vzduchu
        self.air_time = 0

        # Počet možných skoků hráče
        self.jumps = 0

        # Indikátor, zda hráč sklouzává po zdi
        self.wall_slide = False

        # Proměnná pro útok hráče
        self.dashing = 0

        self.fireball_hits = 0
    def update(self, tilemap, movement=(0, 0)):
        """
        Updates the player's state, including movement, jump, dash, and interactions with the game world.

        Parameters:
            tilemap: The tilemap object the player interacts with.
            movement (tuple): The movement input for the player as (dx, dy).
        """
        # Aktualizace stavu hráčovy entity v závislosti na pohybu a kolizích
        super().update(tilemap, movement=movement)

        # Aktualizace doby strávené ve vzduchu
        self.air_time += 1

        if self.air_time > 160:
            self.set_action('hit')
            # Detekce, zda hráč strávil příliš dlouho ve vzduchu
            self.game.sfx['hit'].play()
            self.game.dead =+ 1
            for _ in range(30):  # Generate 20 particles for effect
                angle = random.uniform(0, 2 * 3.14)
                speed = random.uniform(2, 5)
                velocity = [math.cos(angle) * speed, math.sin(angle) * speed]
                self.game.particles.append(Particle(self.game, 'particle', self.rect().center, velocity=velocity

        if self.collisions['down']:
            # Resetování skoků, pokud hráč dosáhl země
            self.air_time = 0
            self.jumps = 1



        self.wall_slide = False
```

```
if (self.collisions['right'] or self.collisions['left']) and self.air_time > 4:
    # Kontrola sklouznutí po zdi a nastavení akce
    self.wall_slide = True
    self.velocity[1] = min(self.velocity[1], 0.5)
    if self.collisions['right']:
        self.flip = False
    else:
        self.flip = True
    self.set_action('wall_slide')

if not self.wall_slide:
    # Nastavení akce podle situace (skok, běh, nečinnost)
    if self.air_time > 4:
        self.set_action('jump')

    elif movement[0] != 0:
        self.set_action('run')
    elif movement[0] == 0:
        self.set_action('idle')

# animace utoku hrace pře ruzne faze
if (self.dashing) >= 55:
    self.set_action('attack')
if (self.dashing) <= -55:
    self.set_action('attack')
if (self.dashing) <= 55:
    if(self.dashing) >= 35:
        self.set_action('attack2')
if (self.dashing) >= -55:
    if(self.dashing) <= -35:
        self.set_action('attack2')
if (self.dashing) <= 35:
    if(self.dashing) >= 25:
        self.set_action('attack3')
if (self.dashing) >= -35:
    if(self.dashing) <= - 25:
        self.set_action('attack3')

if abs(self.dashing) in {70, 60}:
    # Vytvoření efektu střílení (pohybující se částice)
    for i in range(3):
        angle = random.random() * 5 * 2
        speed = random.random() * 1
        pvelocity = [math.cos(angle) * speed, math.sin(angle) * speed]
        self.game.particles.append(Particle(self.game, 'particle', self.rect().center, velocity=pvelocit
if self.dashing > 0:
    self.dashing = max(0, self.dashing - 1)
if self.dashing < 0:
    self.dashing = min(0, self.dashing + 1)
if abs(self.dashing) > 60:
    for i in range(3):
        angle = random.random() * 5 * 2
        speed = random.random() * 1
        pvelocity = [math.cos(angle) * speed, math.sin(angle) * speed]
        self.game.particles.append(Particle(self.game, 'particle', self.rect().center, velocity=pvelocit
    # Nastavení rychlosti hráče při útoku
    self.velocity[0] = abs(self.dashing) / self.dashing * 8

    if abs(self.dashing) <= 61:

        for i in range(3):
            angle = random.random() * 5 * 2
            speed = random.random() * 1
            pvelocity = [math.cos(angle) * speed, math.sin(angle) * speed]
            self.game.particles.append(Particle(self.game, 'particle', self.rect().center, velocity=pvel
        self.velocity[0] *= 0.01
        self.set_action('attack2')




if self.velocity[0] > 0:
    self.velocity[0] = max(self.velocity[0] - 0.1, 0)

else:
    self.velocity[0] = min(self.velocity[0] + 0.1, 0)
```

```
        lethal_decors = tilemap.extract([('large_decor', 2),('large_decor', 3),('large_decor', 4),('large_decor'
        for decor in lethal_decors:
            if self.rect().colliderect(pygame.Rect(decor['pos'][0], decor['pos'][1], tilemap.tile_size, tilemap.
                self.game.sfx['hit'].play()
                self.game.dead =+ 1
                for _ in range(20):  # Generate 20 particles for effect
                    angle = random.uniform(0, 2 * 3)
                    speed = random.uniform(2, 5)
                    velocity = [math.cos(angle) * speed, math.sin(angle) * speed]
                    self.game.particles.append(Particle(self.game, 'particle', self.rect().center, velocity=velc
                    # Or any other logic you use to handle player death


        if self.fireball_hits > 0:
            self.player.set_action('hit')  # Assuming set_action method handles animation change
            self.fireball_hits = 0  # Reset the hit count after playing the animation

    def jump(self):
        """
        Initiates the player's jump, affecting velocity and potentially enabling wall jumps or double jumps.
        """
        # Funkce pro provedení skoku hráče
        if self.wall_slide:
            if self.flip and self.last_movement[0] < 0:
                self.velocity[0] = 3.5
                self.velocity[1] = -2.5
                self.air_time = 5
                self.jumps = max(0, self.jumps - 1)
                return True
            elif not self.flip and self.last_movement[0] > 0:
                self.velocity[0] = -3.5
                self.velocity[1] = -2.5
                self.air_time = 5
                self.jumps = max(0, self.jumps - 1)
                return True

        elif self.jumps:
            self.velocity[1] = -3
            self.jumps -= 1
            self.air_time = 5
            return True

    def dash(self):
        """
        Initiates the player's dash action, providing a burst of speed and potentially passing through obstacles
        """
        # Funkce pro provedení útoku (dash) hráče
        if not self.dashing:
            self.game.sfx['dash'].play()
            if self.flip:
                self.dashing = -70
            else:
                self.dashing = 70

    def rect(self):
        """
        Overrides the base method to provide a rectangle representing the player's bounds for collision detectic

        Returns:
            pygame.Rect: The rectangle representing the player's bounds.
        """
        # Funkce pro získání obdélníkového objektu hráče pro kolizní detekci
        return pygame.Rect(self.pos[0], self.pos[1], self.size[0], self.size[1])
```