

# ZOMBIES

## ALUNOS:

João Neto, a22200558  
Daivd Mendes, a22203255

## INTRODUCTION

Este é um estudo matemático onde tentamos nos provar capazes de implementar mais de 80% da matéria dada em aula em um só jogo, foi desafiante encontrar justificativas de gameplay para tantos conceitos, mas no fim, conseguimos um produto final muito satisfatório, que atende o desafio, fazendo questão de usar cálculos manuais, praticamente sem recursos de bibliotecas, tanto para **vetores**, quanto para **matrizes de transformação**.

---




## HOW TO RUN

Para correr o game é simples, basta executar o comando para o script main.py ou equivalente em outros sistemas operacionais:

```
python main.py
```

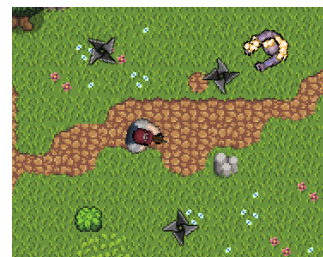
## HOW TO PLAY

Hordas de zombies são guiadas por sua A.I. até o player, onde este pode se defender usando suas armas, a cada zombie morto, 1 ponto é marcado.

- Para rotacionar o player, pode-se usar  e  ou **A** e **D**.
- Para mover o player na direção rotacionada, pode-se utilizar  ou **S**
- Para selecionar uma arma, clique no botão com sua imagem
- Para usar a arma selecionada use a tecla de espaço

## MATRIZ DE ROTAÇÃO

Começando com as matrizes, fizemos uma implementação com estrelas ninjas, onde ao serem lançadas, orbitam o player, elas são levadas até a origem (0, 0), rotacionadas usando a equação da matriz de rotação e retornam para a posição do player:



```
# ROTATING AROUND THE ORIGIN (0,0) USING ROTATION MATRIX OPERATION
# clockwise equation:
#   x' = x * cos(theta) - y * sin(theta)
#   y' = x * sin(theta) + y * cos(theta)
current_position = Vector2(self.DISTANCE_FROM_ORIGIN, self.DISTANCE_FROM_ORIGIN)
new_position = Vector2(0, 0)
new_position.x = current_position.x * math.cos(self.angle) - current_position.y * math.sin(self.angle)
new_position.y = current_position.x * math.sin(self.angle) + current_position.y * math.cos(self.angle)

# MOVES THE ROTATED ITEM BACK TO PLAYER
self.transform.move_position(self.player.transform.get_position_copy() + new_position)
```

## NORMALIZAÇÃO ( $\text{direction} = \text{direction} / \text{numpy.linalg.norm}(\text{direction})$ )

Foi a operação que mais usamos, ela aparecerá 😊, logo abaixo, Player, Zombie e Projéteis, todos utilizam ela.

---

## ÂNGULO VINDO DE DIREÇÃO ( $\text{atan2}(\text{dy}, \text{dx})$ )

Exemplo usado em Zombie para poder rotacioná-lo em direção ao player

```
# ANGLE (used for rotation, atan2 is used to get the angle between two points)
up_direction = Vector2(0, 0)
dx = up_direction.x - self.direction_to_player.x
dy = up_direction.y - self.direction_to_player.y
rads = math.atan2(-dy, dx)
rads %= 2 * math.pi
```

## DIREÇÃO VINDA DE ÂNGULO ( $x = \cos(\theta)$ , $y = \sin(\theta)$ )

Exemplo usado para gerar uma direção para o Player a partir do ângulo que está rotacionado

```
angle_as_radians = (self.angle) * math.pi / 180
self.direction = Vector2(math.cos(angle_as_radians), - math.sin(angle_as_radians))
# normalizes the dir, avoiding div by 0 exception, ex: vector=(0, 0)
if numpy.linalg.norm(self.direction) > 0:
    self.direction = self.direction / numpy.linalg.norm(self.direction)
```

## MOVIMENTO COM MODIFICADOR DE VELOCIDADE (frame-rate independent)

Todos os movimentos e rotações do jogo são feitos com técnicas para de frame-rate independence, como este exemplo de Player:

```
current_position = self.transform.get_position_copy()
new_postion = current_position + self.direction * self.move_speed * GameLoop.Delta_Time
self.transform.move_position(new_postion)
```

## CONVERSÃO (GRAUS $\Leftrightarrow$ RADIANS)

Em várias partes do jogo, pode se ver conversões manuais de radianos para graus e graus para radianos, deixamos em highlight aqui um exemplo e Player e Zombie:

**rad => degree (Player):**

```
angle_as_radians = (self.angle) * math.pi / 180
```

**degree => rad (Zombie):**

```
self.angle_to_player = (rads * 180/math.pi)
```

---

## DISTÂNCIA ENTRE 2 PONTOS (posB - posA)

O Zombie é um exemplo bacana de trigonometria básica, onde para fazê-lo seguir o player, primeiro, tivemos que tirar a distância entre o Zombie e o Player

```
# DISTANCE TO PLAYER (used to get the direction to player)
player_position = self.player.transform.get_position_copy()
zombie_position = self.transform.get_position_copy()
distance_to_player = player_position - self.transform.get_position_copy()
```

## DIREÇÃO DE PONTO A PARA PONTO B (normalize(posB - posA))

Com a distância em mãos, fizemos a direção do zombie apontando para o player, que é basicamente a distância entre os dois normalizada, com isso fizemos o movimento com modificador de velocidade e frame-rate independence.

```
# DIRECTION TO PLAYER (normalizes the dir, avoiding div by 0 exceptions, ex: vector=(0, 0))
self.direction_to_player = distance_to_player
if numpy.linalg.norm(distance_to_player) > 0:
    self.direction_to_player = self.direction_to_player / numpy.linalg.norm(self.direction_to_player)

# MOVEMENT (do not transpass barriers)
new_position = zombie_position + self.direction_to_player * self.move_speed * GameLoop.Delta_Time
self.transform.move_position(new_position)
```

---

## COLISÕES (projeção com dx e dy isolados)

As colisões do jogo foram feitas de forma em que o delta do movimento em x fique separado de y, fazendo assim, um modelo mais fluido de colisões

```
def move_position(self, new_position: Vector2) -> None:
    for other in self.owner.scene.game_objects:
        if other == self.owner:
            continue
        # DX COLLISION
        projection_dx = collider.get_inner_rect_copy()
        projection_dx.centerx = new_position.x
        projection_dx.centery = self.__position.y
        if other.collider_component.is_there_overlap_with_rect(projection_dx):
            new_position.x = self.__position.x
        # DY COLLISION
        projection_dy = collider.get_inner_rect_copy()
        projection_dy.centerx = self.__position.x
        projection_dy.centery = new_position.y
        if other.collider_component.is_there_overlap_with_rect(projection_dy):
            new_position.y = self.__position.y
    self.__position = new_position
```

---

## ENGINE VS JOGO EM MINHA ARQUITETURA

O jogo está separado em pastas, há pastas exclusivas da JNeto Productions Game Engine, essa é uma ferramenta discreta, o jogo em si, é composto por GameObjects, os quais não ficam localizados junto a engine, porém, acessam seus recursos, tais quais, Systems, GameLoop, Components, Scenes, com isso em mente, eu considero tudo aquilo que não está na pasta da engine, como sendo de fato a lógica do game.