APLICAÇÃO DE WAVE FUNCTION COLLAPSE E RAY CASTING NA CRIAÇÃO DE UM JOGO ISOMÉTRICO

Aluno: Thomas Ricardo Reinke

Orientador: Dalton Solano dos Reis

FURB
UNIVERSIDADE DE BLUMENAU

Roteiro

- Introdução
- Objetivos
- Fundamentação teórica
- Trabalhos correlatos
- Descrição da ferramenta
- Resultados
- Conclusões

Introdução

- Desenvolvimento dos computadores e aumento do poder de processamento.
- Gráficos 2D, isométricos e 3D.
- Geração de conteúdo por meio de algoritmos (Procedural Content Generation).
- Introdução do algoritmo Wave Function Collapse na área tecnológica recentemente, possuindo um grande potencial no campo do desenvolvimento de jogos.

Objetivos

- Disponibilizar um ambiente isométrico gerado pelo algoritmo de WFC com a aplicação do algoritmo de Ray Casting para simulação do campo de visão.

- Os objetivos específicos são:
  - demonstrar a utilização do algoritmo de WFC com a adição de restrições;
  - demonstrar a utilização do algoritmo de Ray Casting;
  - comparar as performances dos diferentes algoritmos de Ray Casting.

Fundamentação Teórica

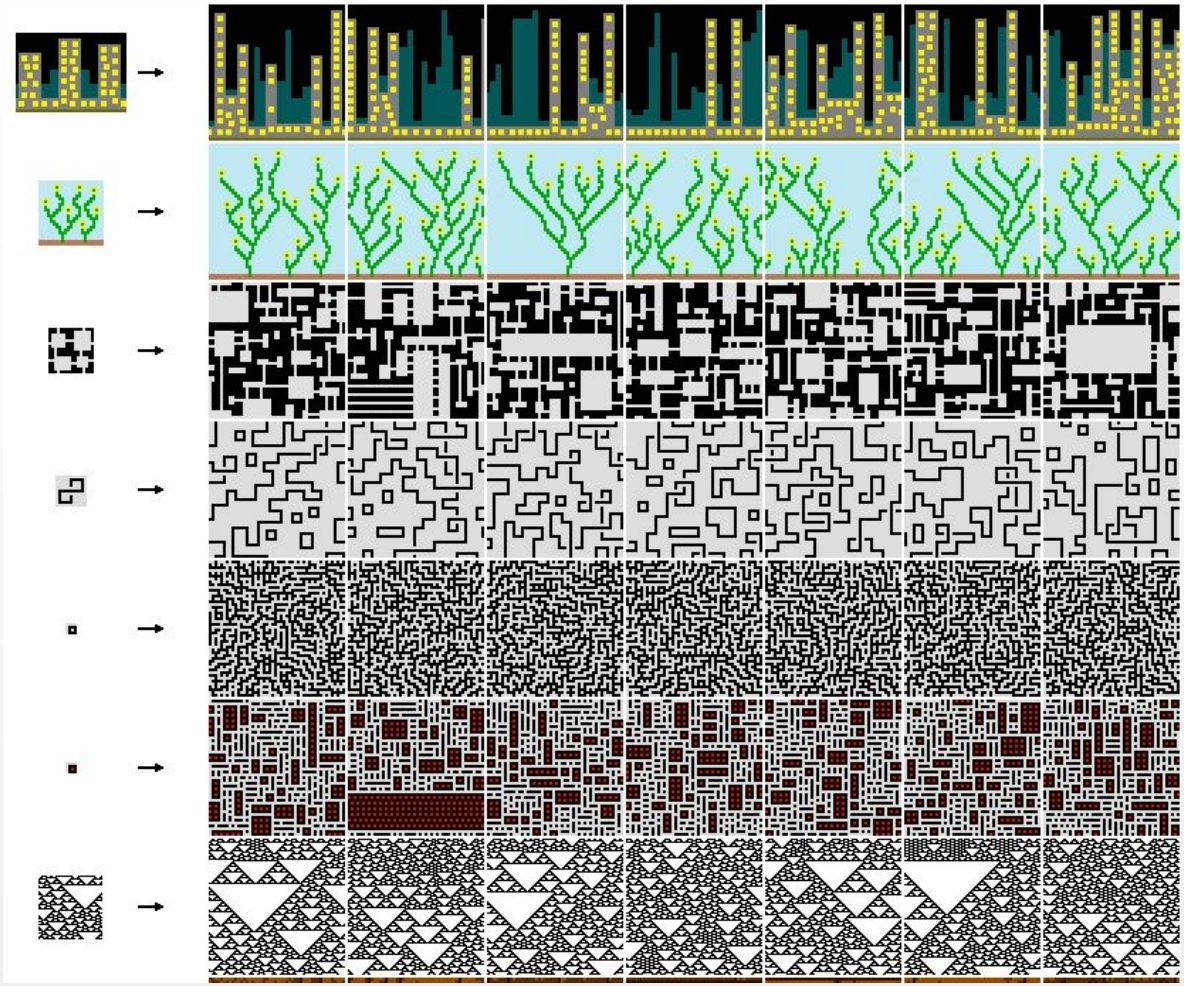
Fundamentação Teórica

Wave Function Collapse

- O WFC é um algoritmo de satisfação de restrição inspirado no conceito do colapso da função de onda presente na física quântica.
- Um estado não observado possibilita que todos os estados sejam possíveis, enquanto observações restringem as possibilidades.
- Até então, era só utilizado para geração de imagens.

Fundamentação Teórica

Wave Function Collapse



Fundamentação Teórica

Ray Casting

- O Ray Casting foi utilizado primeiramente como uma técnica de renderização 3D partindo de uma matriz 2D.

- Consiste de uma quantidade definida de raios saindo de um ponto em comum, e percorrem a matriz até encontrarem algum obstáculo.

- São possíveis diversas abordagens para realizar o cálculo do algoritmo.

Fundamentação Teórica

Ray Casting

Fundamentação Teórica

Isometria

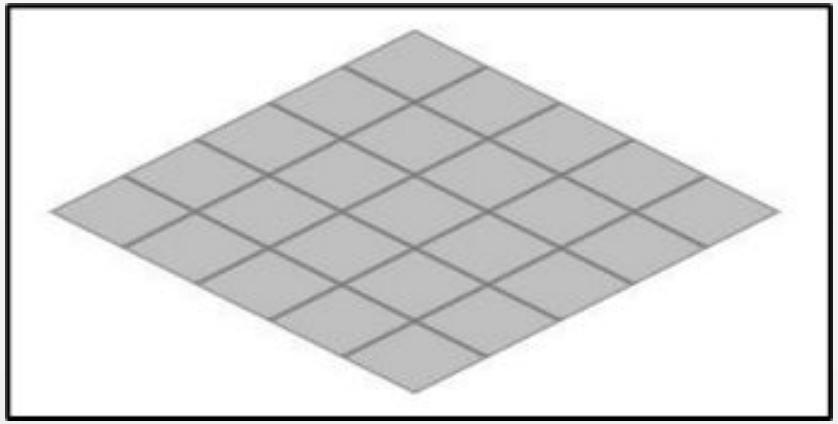
- Projeções isométricas são projeções do espaço 3D para o 2D que possuem as seguintes características:
  - a projeção no espaço 2D não possui “ponto de fuga”;
  - linhas paralelas no espaço 3D continuam paralelas no espaço 2D;
  - objetos que estão distantes possuem o mesmo tamanho de objetos que estão perto;
  - As medidas utilizadas nos eixos x, y e z são as mesmas.

Fundamentação Teórica

Isometria

- *Tiles* precisam ser renderizados do fundo para frete.

- Há várias formas de construir um mapa isométrico, sendo um dos principais, o formato *diamond*, geralmente utilizado em jogos de estratégia.



Fundamentação Teórica

Isometria

Trabalhos Correlatos (1/3)

Enhancing wave function collapse with design-level constraints
Sandhu, Chen e McCoy (2019)

- Implementam diversas restrições na geração do resultado do algoritmo WFC. Como *non-local constraints*, *weight recalculation* e *area propagation*.

- Os testes apresentam uma eficácia suficiente para utilizar as restrições em tempo de execução.

Trabalhos Correlatos (2/3)

Augmented reality in cultural heritage: Field of view awareness in an archaeological site mobile guide
Kasapakis, Gavalas e Galatis (2016)

- É aplicado o algoritmo de Ray Casting juntamente com a realidade aumentada para produzir um aplicativo de pontos de interesse (POIs).

- Os testes foram realizados com diferentes ângulos e comprimentos de *rays*, concluindo que há possibilidade de utilizar o algoritmo em tempo real.

Trabalhos Correlatos (3/3)

Forge 16V: um framework para o desenvolvimento de jogos isométricos
Sampaio e Ramalho (2003)

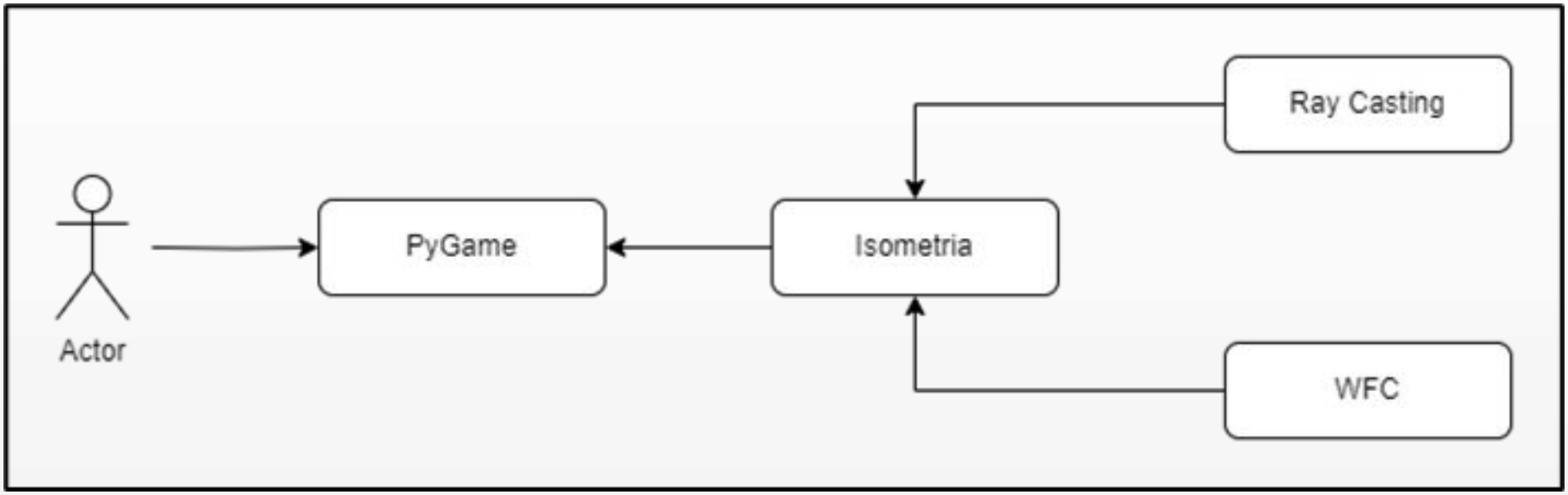
- Foi desenvolvido um framework composto por módulos gráficos, entrada, saída, som, log, IA, modelagem e editor de cenários, para o desenvolvimento de jogos isométricos.

- Os resultados foram medidos com base na validação do framework no curso de Graduação em Ciência da Computação do Centro de Informática da UFPE. Considerado de fácil utilização, robusto e útil.

Descrição do Protótipo

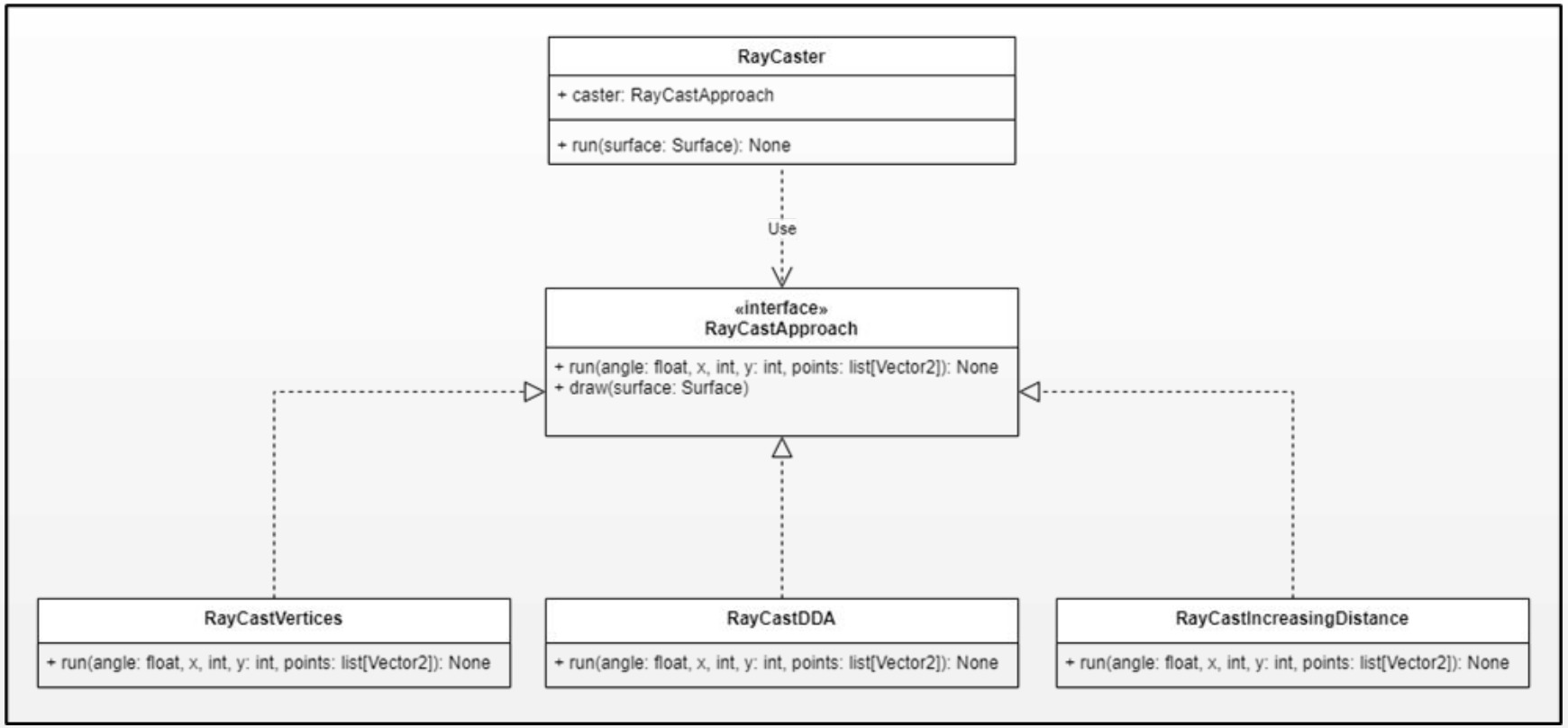
Especificação

Fluxo do funcionamento dos algoritmos



Especificação

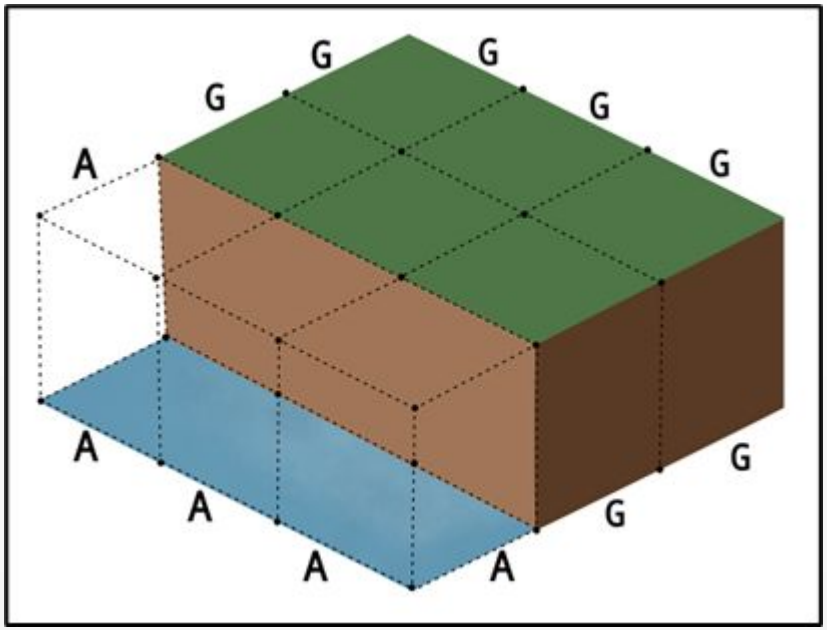
Diagrama de classes dos algoritmos de Ray Casting



Implementação

Implementação

Wave Function Collapse



Implementação

Wave Function Collapse

```
self.file_lookup_table = {
    "water_bottom": ("GGG", "GGA", "AAA", "AGG"),
    "water_right": ("GGA", "AAA", "AGG", "GGG"),
    "water_left": ("AGG", "GGG", "GGA", "AAA"),
    "water_top": ("AAA", "AGG", "GGG", "GGA"),
    "water_top_right": ("AAA", "AAA", "AGG", "GGA"),
    "water_top_left": ("AAA", "AGG", "GGA", "AAA"),
    "water_bottom_right": ("GGA", "AAA", "AAA", "AGG"),
    "water_bottom_left": ("AGG", "GGA", "AAA", "AAA"),
    "grass_bottom_left": ("GGA", "AGG", "GGG", "GGG"),
    "grass_bottom_right": ("AGG", "GGG", "GGG", "GGA"),
    "grass_top_left": ("GGG", "GGA", "AGG", "GGG"),
    "grass_top_right": ("GGG", "GGG", "GGA", "AGG"),
    "grass_stone_bottom": ("GGG", "GGG", "PPP", "GGG"),
    "grass_stone_left": ("GGG", "GGG", "GGG", "PPP"),
    "grass_stone_right": ("GGG", "PPP", "GGG", "GGG"),
    "grass_stone_up": ("PPP", "GGG", "GGG", "GGG"),
    "grass": ("GGG", "GGG", "GGG", "GGG"),
    "water": ("AAA", "AAA", "AAA", "AAA"),
    "stone": ("PPP", "PPP", "PPP", "PPP"),
}
```

Implementação

Wave Function Collapse

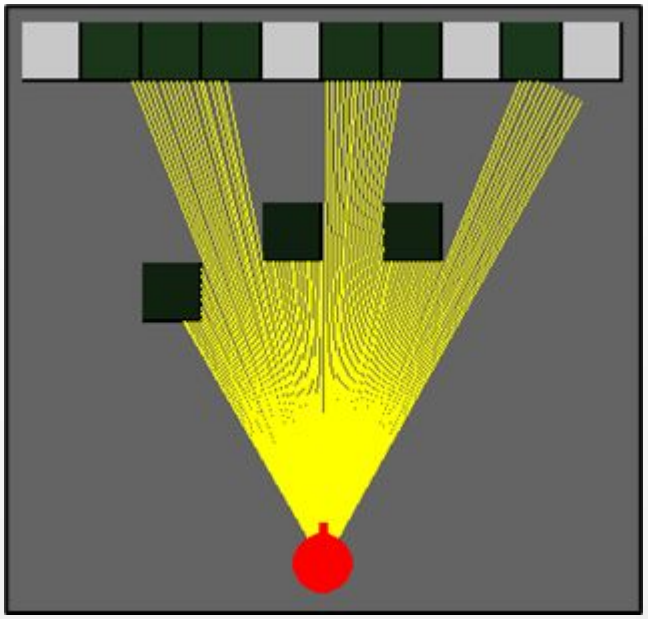
```
if (x + 1, y) in self.filled_set:
    ref = self.board[y][x + 1]
    if ref is not None:
        ref = ref.left[::-1]
        localset = {tile for tile in self.tiles if self.tiles[tile].right == ref}
else:
    localset = set(self.tiles)

outset &= localset
```

```
weights = self.get_weights(entropy)
entropy = choices(entropy, weights)
```

Implementação

Ray Casting (Pixel)

```
start_angle = player_angle – (self.fov / 2)

for ray in range(self.rays_to_cast):
    target_sin = math.sin(start_angle)
    target_cos = math.cos(start_angle)

    for depth in range(self.max_depth):
        target = Vector2(player_x - target_sin *
depth, player_y + target_cos * depth)
```



Implementação

Ray Casting (Digital Differential Analyzer)

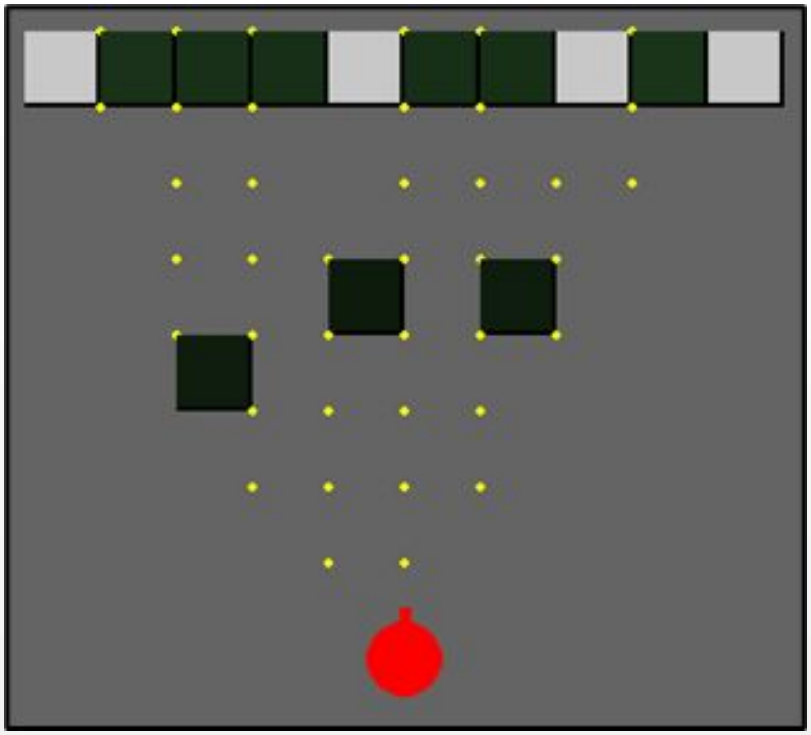
```
player_in_map = Vector2(player_x // self.map.tile_size, player_y //
self.map.tile_size)

normalized_direction = (target - player).normalize()
normalized_direction.x -= 0.0000001
normalized_direction.y -= 0.0000001

ray_stepsize = Vector2(
    math.sqrt(1 + math.pow((normalized_direction.y / normalized_direction.x), 2)),
    math.sqrt(1 + math.pow((normalized_direction.x / normalized_direction.y), 2))
)
```

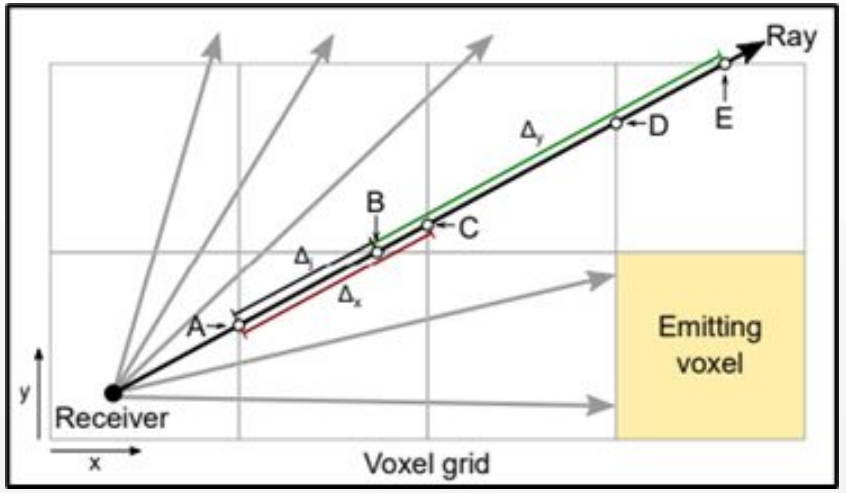
$$Sx \leftarrow \sqrt{1 + \left(\frac{dy}{dx}\right)^2}$$

$$Sy \leftarrow \sqrt{1 + \left(\frac{dx}{dy}\right)^2}$$

Implementação

Ray Casting (Digital Differential Analyzer)





Implementação

Ray Casting (Vértice)

```
unique_angles: list[float] = []
for point in unique_points:
    angle = math.atan2(point[0].y - player_y, point[0].x - player_x)
    unique_angles.append(angle - 0.00001)
    unique_angles.append(angle)
    unique_angles.append(angle + 0.00001)

r_mag = math.sqrt(r_dx * r_dx + r_dy * r_dy)
s_mag = math.sqrt(s_dx * s_dx + s_dy * s_dy)
if r_dx / r_mag == s_dx / s_mag and r_dy / r_mag == s_dy / s_mag:
    return None
```

$$r\_px + r\_dx * T1 = s\_px + s\_dx * T2$$

$$r\_py + r\_dy * T1 = s\_py + s\_dy * T2$$

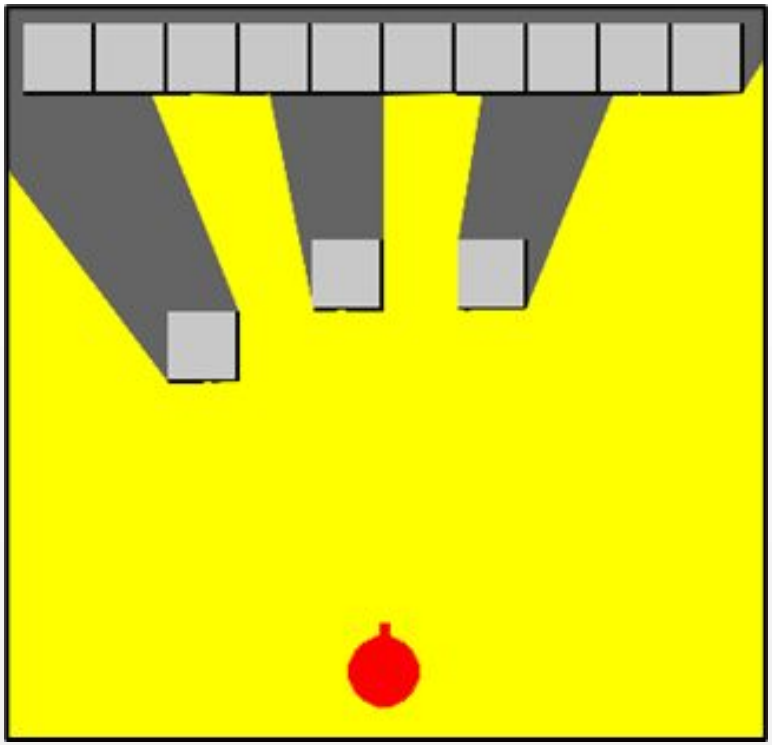
Implementação

Ray Casting (Vértice)

```
try:
    T2 = (r_dx * (s_py - r_py) + r_dy * (r_px - s_px)) / (s_dx * r_dy - s_dy * r_dx)
except ZeroDivisionError:
    T2 = (r_dx * (s_py - r_py) + r_dy * (r_px - s_px)) / (s_dx * r_dy - s_dy * r_dx -
0.00001)

try:
    T1 = (s_px + s_dx * T2 - r_px) / r_dx
except ZeroDivisionError:
    T1 = (s_px + s_dx * T2 - r_px) / (r_dx - 0.00001)

if T1 < 0: return None
if T2 < 0 or T2 > 1: return None

return {
    "x": r_px + r_dx * T1,
    "y": r_py + r_dy * T1,
    "param": T1
}
```

Implementação

Ray Casting (Vértice)



Implementação

Isometria

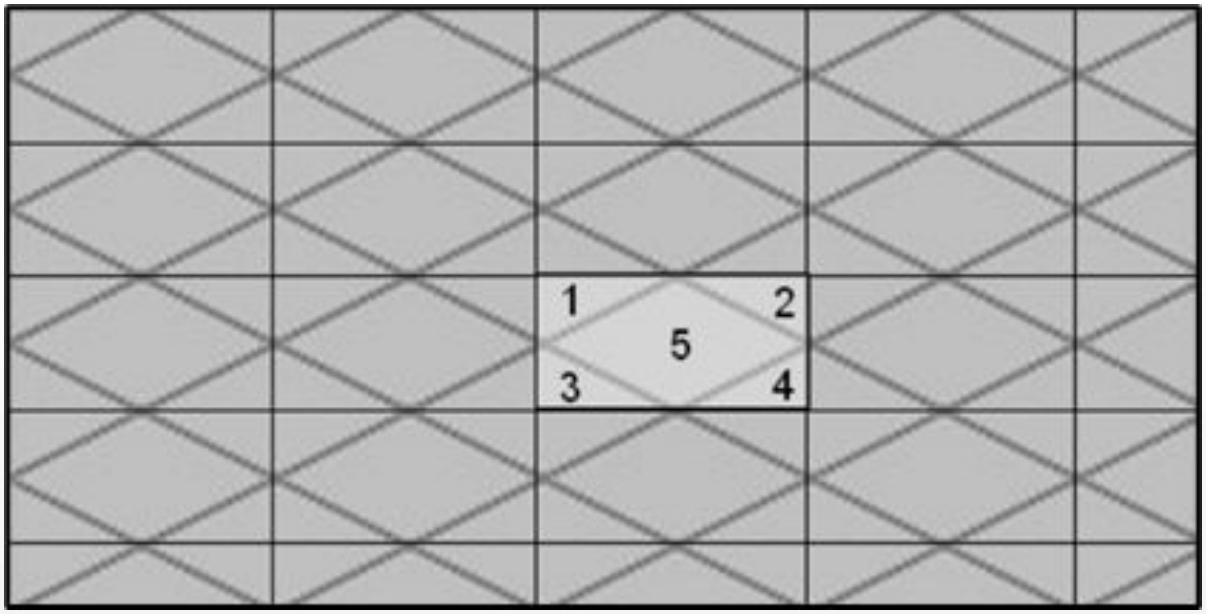
```
def to_screen_coordinates(x: int, y: int, offset: Vector2):
    return Vector2(
        (VECTOR_ORIGIN.x*VECTOR_TILESIZE.x)+(x-y)*(VECTOR_TILESIZE.x/2)+offset.x,
        (VECTOR_ORIGIN.y*VECTOR_TILESIZE.y)+(x+y)*(VECTOR_TILESIZE.y/2)+offset.y,
    )

def to_list_coordinates(x: int, y: int):
    return Vector2(
        int((y - VECTOR_ORIGIN.y) + (x - VECTOR_ORIGIN.x)),
        int((y - VECTOR_ORIGIN.y) - (x - VECTOR_ORIGIN.x))
    )
```

$$Sx \leftarrow (x - y) \cdot \frac{Tw}{2}$$

$$Sy \leftarrow (x + y) \cdot \frac{Th}{2}$$

Implementação

Isometria



Implementação

Isometria

```
def sign(p1: Vector2, p2: Vector2, p3: Vector2):
    return (p1.x - p3.x) * (p2.y - p3.y) - (p2.x - p3.x) * (p1.y - p3.y)

def is_point_inside(point: Vector2, v_a: Vector2, v_b: Vector2, v_c: Vector2):

    d1 = sign(point, vertice_a, vertice_b)
    d2 = sign(point, vertice_b, vertice_c)
    d3 = sign(point, vertice_c, vertice_a)

    has_neg = (d1 < 0) or (d2 < 0) or (d3 < 0)
    has_pos = (d1 > 0) or (d2 > 0) or (d3 > 0)

    return not (has_neg and has_pos)
```

Análise dos Resultados

Wave Function Collapse

Dimensões	Qtd. *Tiles*	ms
5x5	25	0,8
10x10	100	7,2
25x25	625	223,4
50x50	2.500	3.443,2
75x75	5.625	15.111,0
100x100	10.000	119.395,9

Análise dos Resultados

Ray Casting (Pixel e DDA)

Algoritmo	Qtd *rays*	Distância máxima	ms
Pixel	150	300	31,8
	150	450	51,3
	100	300	28,5
	100	450	38,8
	50	300	13,8
	50	450	17,5
DDA	150	300	2,4
	150	450	5,1
	100	300	2,2
	100	450	3,5
	50	300	1,3
	50	450	1,6

Análise dos Resultados

Ray Casting (Vértices)

Qtd vértices	ms
20	2,5
40	6,6
80	21,0
120	45,2
160	86,0

Conclusões

- O objetivo de demonstrar como os algoritmos de WFC e Ray Casting podem ser utilizados para gerar uma projeção isométrica em um jogo foi alcançado.
- A abordagem para calcular o algoritmo de Ray Casting é determinada com base no objetivo do desenvolvedor.
- O algoritmo WFC é mais performático em gerações menores.

Conclusões

- A utilização do *weighted choice* para seleção de *tiles* no algoritmo WFC proporcionou uma maior naturalidade nos resultados.

Sugestões de Melhoria

- utilizar diferentes abordagens de cálculo para o algoritmo de Ray Casting;
- criar um algoritmo para reduzir a quantidade de pontos calculados pela abordagem do Ray Casting que utiliza vértices, realizando uma espécie de convex-hull nos pontos para obter apenas os pontos exteriores de cada figura;
- incluir diferentes camadas de solo à geração do algoritmo de WFC, possibilitando que sejam gerados terrenos isométricos com diferentes alturas;

Sugestões de Melhoria

- utilizar os algoritmos de Ray Casting e WFC para a geração de ambientes com projeções diferentes da isométrica, sendo elas 2D ou até mesmo 3D, como descrito por Vandevenne (2004);
- implementar o backtracking ao algoritmo WFC conforme proposto por Gumin (2019) e por Sandhu et al. (2019), para realizar a reconstrução em caso de impossibilidade de continuação;
- gerar pequenos novos chunks de forma procedural, conforme o usuário se aproxima dos limites do terreno já gerado, para possibilitar uma execução mais rápida do algoritmo;

Apresentação Prática

- Abordagens do algoritmo Ray Casting.
- Projeto final.