PONTIFÍCIA UNIVERSIDADE CATÓLICA DE GOIÁS
ESCOLA POLITÉCNICA
GRADUAÇÃO EM CIENCIAS DA COMPUTAÇÃO

Criação de mapas utilizando geração procedural

GUILHERME NEVES CANEDO

GOIÂNIA
2022

PONTIFÍCIA UNIVERSIDADE CATÓLICA DE GOIÁS
ESCOLA POLITÉCNICA
GRADUAÇÃO EM CIENCIAS DA COMPUTAÇÃO

GUILHERME NEVES CANEDO

CRIAÇÃO DE MAPAS UTILIZANDO GERAÇÃO PROCEDURAL

Trabalho de conclusão de curso apresentado a escola de ciências exatas e da computação da Pontifica Universidade Católica de Goiás como parte dos requisitos para obtenção do título de bacharel em ciências da computação.
Orientador: Gustavo Siqueira Vinhal

GOIÂNIA
2022

GUILHERME NEVES CANEDO

CRIAÇÃO DE MAPAS UTILIZANDO GERAÇÃO PROCEDURAL

Este trabalho de conclusão de curso foi julgado adequado para obtenção do título de bacharel em ciência da computação e aprovado em sua forma final pela escola politécnica da Pontifica Universidade Católica de Goiás em ____/____/_____.

_____

Prof. Ma. Ludmila Reis Pinheiro dos Santos
Coordenadora do trabalho de conclusão de curso

Banca examinadora:

_____

Prof. Me. Gustavo Siqueira Vinhal

_____

Prof. Me. Rafael Leal Martins

_____

Prof. Me. Fernando Gonçalves Abadia

GOIÂNIA
2022

RESUMO

O desenvolvimento de jogos está se tornando mais complexo. Conforme isso acontece, cada vez mais recursos são necessários para garantir a qualidade. Com o aumento dos custos nem todos os desenvolvedores possuem os recursos necessários para criar grandes jogos. Um dos métodos que podem ser utilizados para aliviar os custos de produção é a utilização da geração procedural de conteúdo. Para a utilização do GPC é utilizado algo chamado de ruido. Ruido é a atribuição de valores aleatórios em uma matriz e os valores contidos nessa matriz são utilizados para gerar elementos como terreno, a posição de itens em um mapa, o posicionamento da vegetação em um ambiente, entre outras possibilidades. O resultado obtido com esse trabalho não é suficiente para ser utilizado em um jogo eletrônico da maneira em que está, mas existem recomendações sobre como melhorá-lo e adaptá-lo para que possa ser utilizado de maneira mais eficiente.

Palavras-chave: Geração procedural. Ruído. Jogos eletrônicos.

ABSTRACT

Developing games is getting more and more complex. With that, more resources are necessary to create quality games. With the rising costs not all developers have the resources to create big games. One of the methods that can be used to alleviate production costs is the use of procedural content generation. To use PCG we need something called noise. Noise is the attribution of random values in a matrix, and the values contained in this matrix are used to generate elements such as terrain, the position of items on a map, the positioning of vegetation in an environment, among other possibilities. The result obtained from this work is not enough to be used in an electronic game the way it is, but there are recommendations on how to improve and adapt it so that it can be used more efficiently.
Keywords: Procedural generation. Noise. Electronic games.

LISTA DE IMAGENS

SUMARIO

1. INTRODUÇÃO

A indústria dos jogos eletrônicos tem se tornado cada vez mais popular e rentável. Segundo uma reportagem publicada por *What Box Gamin,* o jogo *God of War Ragnarok*, publicado em 9 de novembro de 2022, teve um custo de produção de aproximadamente 500 milhões de dólares.

Conforme a complexidade de um jogo aumenta, o seu custo de produção também aumenta. Os custos da área tecnológica podem ser aliviados ao utilizar tecnologias licenciadas por terceiros como, por exemplo, *middlewares* específicos para gráficos, física e inteligência artificial. No entanto, não existe um facilitador desse tipo para as equipes de arte e design. Os elementos de jogo, como personagens, história, terrenos, missões, itens, inimigos, objetos, mapas, devem ser feitos individualmente. Isto aumenta o custo de produção e o tempo de desenvolvimento de maneira considerável (DUARTE, 2012).

Uma das soluções utilizadas para reduzir o custo do desenvolvimento é a criação algorítmica de conteúdo para o jogo: a Geração Procedural de Conteúdo (GCP) (BALDWIN, DAHLSKOG, *et al.*, 2017).

A GPC refere-se à criação algorítmica de conteúdo. Isso permite que o conteúdo seja gerado automaticamente e pode diminuir significativamente a carga de trabalho dos artistas (LINDEN, LOPES e BIDARRA, 2014). A geração procedural já vem sendo utilizada em diversas áreas para geração de, por exemplo, estampas para roupa, *bots* que geram textos para serem postados nas redes sociais, poesia, música e até histórias. Em jogos, a GPC é utilizada para geração de vegetação, itens, criaturas que habitam o mundo, terrenos e mapas (COMPTON, 2017).

A GPC já é utilizada por várias desenvolvedoras para acelerar o processo de produção de jogos e facilitar o trabalho de artistas, possibilitando resultados de melhor qualidade em tempos menores. Logo, o estudo dessa área é importante para os desenvolvedores de jogos, principalmente para estúdios que não possuem grandes recursos financeiros.

1.1 Objetivo geral

Demonstrar um método de geração de mapa que pode ser utilizado em um jogo eletrônico.

1.2 Objetivos específicos

- Exemplificar métodos de criação de Ruído;
- Exemplificar o uso da geração procedural em jogos;
- Demonstrar a geração de um mapa de Ruído (*noise map)*;
- Demonstrar como o Ruído pode ser utilizado para geração de um mapa.

1.3 Resultados esperados

Com este trabalho tem-se o desejo de utilizar os métodos estudados para a geração de um mapa em 2D ou 3D que possa ser utilizado em um jogo eletrônico.

1.4 Justificativa

Este trabalho tem como justificativa a exemplificação de métodos de geração procedural e a demonstração de como podem ser produzidos para que seja um facilitador para desenvolvedores que desejam produzir algo semelhante. A importância desse estudo se concentra em pequenos desenvolvedores que não possuem orçamento suficiente para desenvolver seus jogos.

1.5 Estrutura do trabalho

No Capítulo 2 é apresentado toda a fundamentação teórica necessária para o entendimento do trabalho. No Capítulo 3 é explicado o processo de criação do algoritmo desenvolvido, assim como as ferramentas utilizadas. O Capítulo 4 apresenta os resultados obtidos com o algoritmo e seu funcionamento. Por último, no Capítulo 5, são mostradas as conclusões e sugestões de trabalhos futuros.

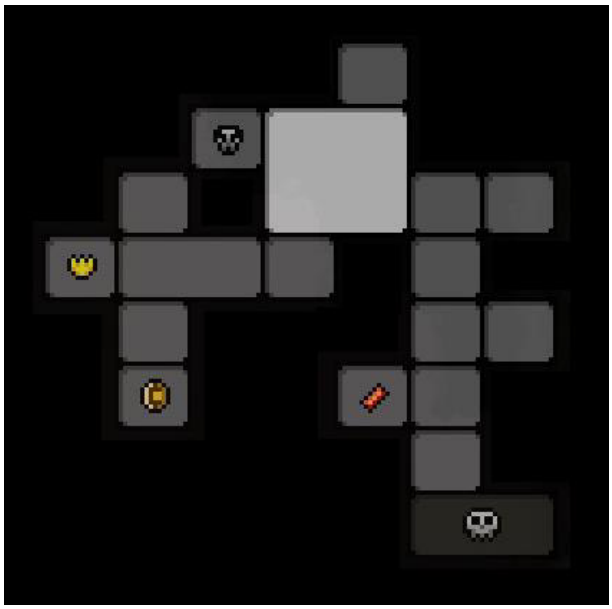
2. REFERENCIAL TEORICO

2.1 Geração aleatória *versus* geração procedural

Geração aleatória e geração procedural são dois métodos para geração de conteúdo aleatório. A geração aleatória consiste na inserção aleatória de elementos individuais que foram criados anteriormente. Esse método é utilizado, por exemplo, no jogo *The binding of isaac.* Os elementos do jogo como itens, salas e inimigos são apenas inseridos de maneira aleatória, formando o mapa do jogo (DIGIDIGGER, 2020).

A Figura 1 apresenta um mapa o jogo *The Binding of Isaac*. No centro é criado a sala inicial, onde o personagem aparecerá quando entrar no level. O número de salas que terá nesse level é escolhido e em seguida são geradas as salas. Primeiramente são geradas as salas normais e no máximo 5 dessas não terão continuação (chamados pelo jogo de *dead end*). Nessas salas são adicionadas as salas especiais. Uma delas será a sala do chefe (representada pelo símbolo de caveira), um quarto supersecreto que não é mostrado no mapa e deve ser descoberto pelo jogador, e outros que podem incluir um mercado, uma sala de tesouro, salas com mini chefes, salas de desafio, entre outros (BINDING OF ISAAC WIKI, [20--]).

Figura 1: Exemplo de mapa presente no jogo *The binding of Isaac*



Fonte*:* (BORIS,2020)

A geração procedural é feita de maneira diferente. Na geração procedural todo o jogo, ou boa parte dele, é gerado por um algoritmo. Isso aumenta a variedade na distribuição dos elementos (vegetação, itens, entre outros) e dependendo de como é feito pode gerar um valor “infinito” de possibilidades.

Um exemplo de jogo que utiliza esse método é *Minecraft.* Cada vez que se inicia um novo jogo, o algoritmo de geração procedural é executado e um novo mundo é criado a partir de variáveis. Para aumentar a variedade cada tipo de elemento é gerado com um novo mapa de ruido. (MINECRAFT WIKI, [20--]).

Primeiramente são gerados e definidos os biomas (Floresta, deserto, pântano etc.). Em seguida, com um mapa de ruido diferente, são gerados os decoradores (Flores, arvores, animais). Também é utilizado ruido para a geração de cavernas, ravinas, masmorras etc. Os valores utilizados para a geração são salvos na *seed.* Caso o jogador queira, ele pode inserir um valor de *seed* específico. Isso pode ser utilizado caso ele queira iniciar em um bioma específico ou algo semelhante. Na Figura 2 é mostrado o resultado da *seed* -1834063422, que inicia o player em uma vila abandonada no deserto. (MINECRAFT WIKI, [20--]).

Figura 2:Vila abandonada no jogo *Minecraft.*

Fonte: (MINECRAFT WIKI, [20--])

2.2 *Ruído*

Na GPC, Ruído (*Noise*) é a geração de números aleatórios graficamente. Os valores de cada pixel são gerados de maneira aleatória e não estruturada. Ao se gerar

algo proceduralmente, o ruido é a fonte dos valores aleatórios que serão utilizados para definir, por exemplo, que tipo de ambiente será gerado em cada ponto do mapa ou se naquele ponto em específico deveria ser adicionado uma arvore ou um item. As características do ruído são definidas de maneira semelhante a ondas sonoras na música ou na física, onde apresenta frequência, amplitude, oitavas, lacunaridade, persistência (LAGAE, LEFEBVRE, *et al.*, 2010).

Existem várias funções diferentes para geração de ruído. Cada uma delas apresenta características diferentes em seu funcionamento e podem ser usadas para gerações de objetos diferentes. Alguns exemplos são o *Perlin noise,* o *Wavelet noise* e o *Spot noise*.

2.3 Elementos do *noise*

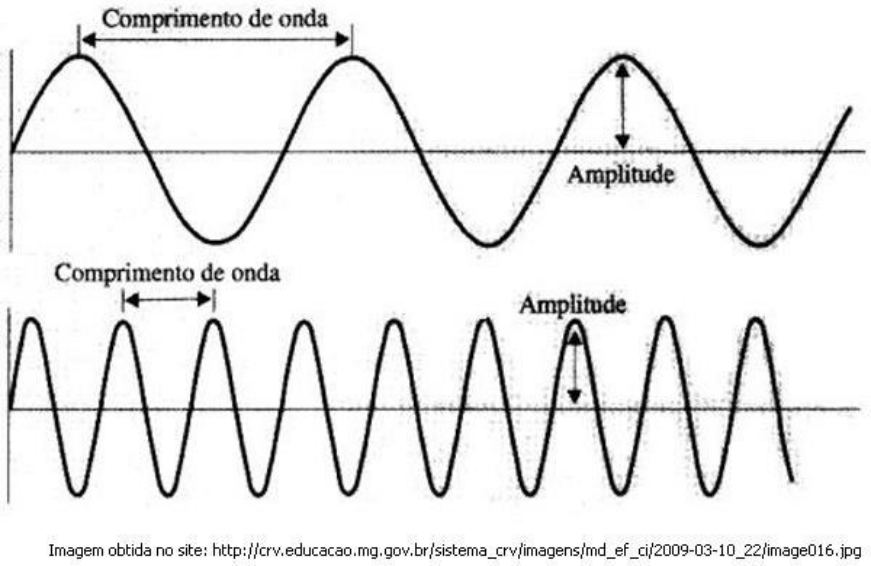
Para aumentar a variação na geração foi utilizado também a frequência, amplitude, oitavas, lacunaridade e persistência.

A frequência funciona de maneira igual a quando se refere a uma onda. A frequência causa variações no eixo x, aumentando a velocidade em que os valores podem aumentar e diminuir.

A amplitude também é um termo existente quando se refere a ondas. A amplitude causa variações no eixo y. Conforme a amplitude aumenta os valores máximos e mínimos obtidos também aumentam.

A Figura 3 ilustra a amplitude e frequência de uma onda. Na parte superior da imagem é representado uma onda com uma frequência menor do que a imagem inferior. A amplitude representa a altura máxima da onda.

Figura 3: Ilustração da frequência e amplitude de uma onda



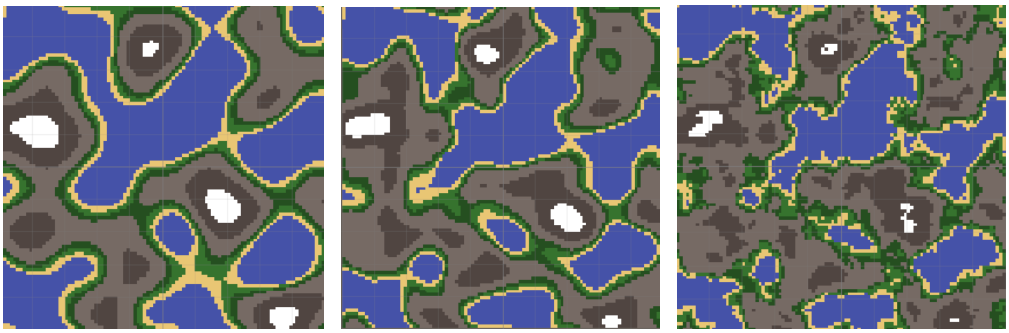
Fonte: (FONETICA E FONOLOGIA, 2008)

As oitavas são quantas camadas de terreno serão sobrepostas para gerar o resultado. Cada camada será afetada pela lacunaridade e pela persistência de maneira mais intensa, causando variações mais suaves no terreno e causando um resultado mais realista.

A lacunaridade irá controlar o aumento da frequência à cada oitava. Já a persistência irá controlar a diminuição da amplitude. Ou seja, a cada oitava a frequência se torna maior e a amplitude se torna menor. Assim, a primeira oitava irá definir o formato geral do terreno, com as montanhas e depressões; a segunda oitava irá amenizar o terreno, fazendo com que as mudanças aconteçam mais suavemente; e a terceira irá causar mais variações nos terrenos baixos, por exemplo.

A Figura 4 demonstra como a lacunaridade e persistência afetam o mesmo terreno quando se tem mais oitavas. O terreno à esquerda apresenta apenas uma oitava. O do meio duas oitavas. O da direita seis oitavas.

Figura 4: Geração do mesmo terreno, mas com cada vez mais oitavas.

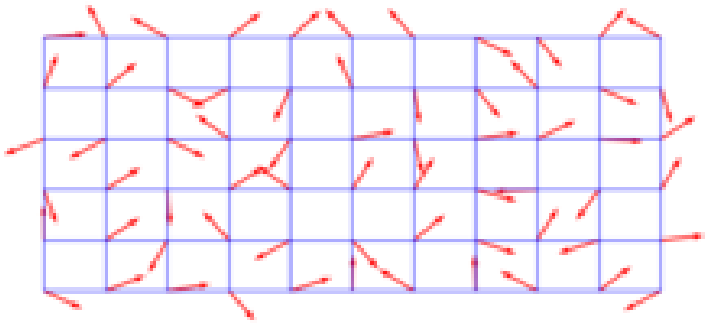


Fonte: Autor

2.3 *Perlin Noise*

Criado em 1985 por Perlin é um dos ruídos mais conhecidos e utilizados. O ruído em um ponto é definido a partir da computação de um gradiente pseudoaleatório em cada um de seus vértices. Na Figura 5 pode ser visualizado esse gradiente e seus vetores. Quanto mais próximo da ponta do vetor, mais próximo o resultado está de 1, e quanto mais distante mais próximo de 0.

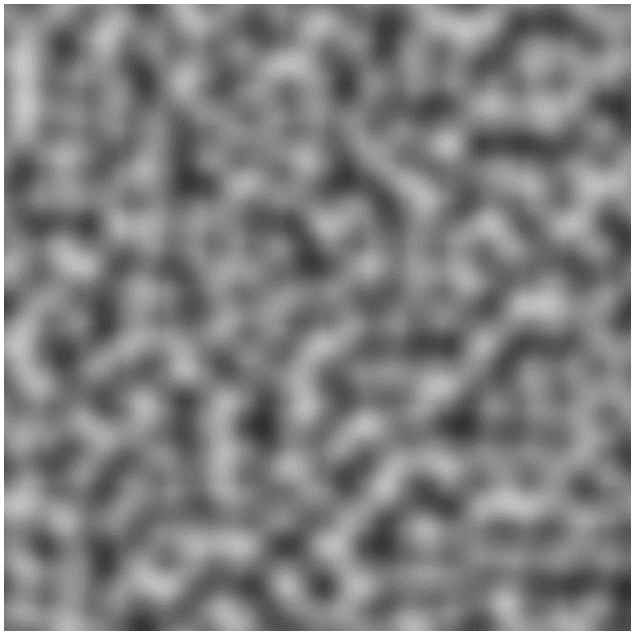
Figura 5: Ilustração do gradiente utilizado no algoritmo de Perlin.



Fonte: (WIKIPEDIA, [20--]

O resultado são valores entre 0 e 1 atribuídos a coordenadas em um mapa de ruído como ilustrado pela Figura 6. O mapa de ruido é a demonstração gráfica desses valores em uma escala de cinza para que sejam mais bem visualizados. Uma coordenada com valor igual a 0 tem sua posição no mapa pintada de preto. Conforme os valores aumentam, sua coloração se torna mais clara até se tornar branca quando tem valor 1. *Perlin Noise* é amplamente utilizada na computação gráfica por ser rápido e simples. (LAGAE, LEFEBVRE, *et al.*, 2010).

Figura 6: mapa de Ruído de Perlin.



Fonte: (WIKIPEDIA, [20--]

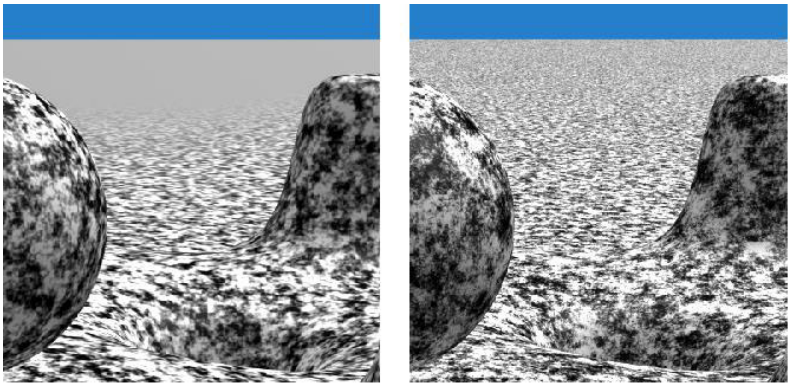
Apesar de ser muito utilizado na computação gráfica, ele apresenta um problema quando é utilizado em coordenadas muito distantes. A perda de qualidade se torna tão grande que se torna quase impossível de perceber diferenças nos valores de ruido. Por isso, em 2005, a *Pixar Animation* desenvolveu uma melhoria desse método para melhorar a qualidade de suas animações, o *Wavelet noise*.

2.4 *Wavelet Noise*

Criado em 2005 por Cook e DeRose ao perceberem o problema de serrilhamento e perda de detalhes do *Perlin noise*. Sua geração é feita a partir do *Perlin noise* mas, devido ao tratamento feito, consegue manter a qualidade de imagem maior em distâncias maiores (COOK e DEROSE, 2005).

A Figura 7 ilustra o ruido desenvolvido. É possível perceber que na imagem da esquerda a qualidade diminui em distancias maiores. Problema que não é percebido na imagem da direita.

Figura 7: A mesma geração utilizando *perlin noise*(esqueda) e *wavelet noise*(direita).

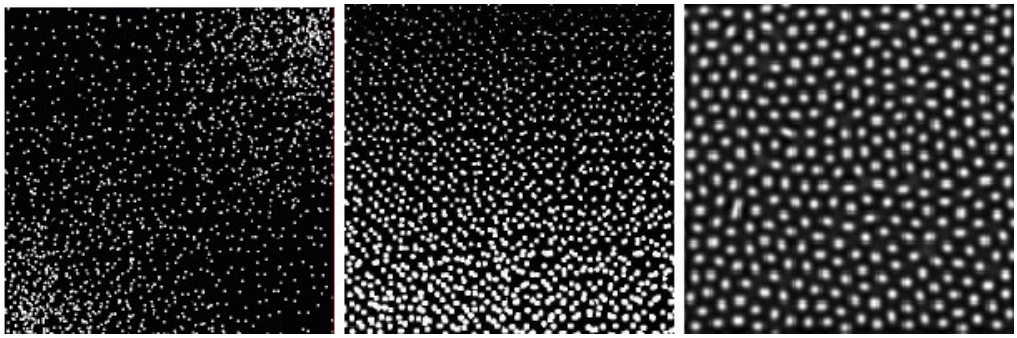


Fonte: (COOK e DEROSE, 2005)

2.5 *Spot noise*

O *spot noise* é gerado através da adição de pontos com posições e pesos aleatórios. A textura é variada com a mudança dos pontos. É muito efetivo na texturização de superfícies curvas e pode ser utilizado para visualização de informações, como escalares e vetores, em uma superfície (WIJK, 1991). A Figura 8 ilustra diferentes exemplos de *spot noise*.

Figura 8: renderização de diferentes *spot noises*.



Fonte: (HOTZ, 2009)

3. PROCEDIMENTO METODOLOGICO

Neste capítulo será explicado os métodos e algoritmos utilizados. O algoritmo de geração do *noise*, o desenho das texturas, a formação dos *meshes* e as ferramentas utilizadas.

3.1 Geração do *noise*

O ruído de Perlin foi utilizado para o desenvolvimento deste trabalho, por ser o mais utilizado para geração de terrenos.

O algoritmo é implementado como uma função de duas, três ou quatro dimensões, mas pode ser utilizado para qualquer número de dimensões. Sua implementação envolve a definição de um gradiente aleatório, a computação do produto entre esse gradiente e seus *offsets* e por último a interpolação desses valores. (LAGAE, LEFEBVRE, *et al.*, 2010)

Várias ferramentas já possuem uma função que cria o *Perlin noise* sem que seja necessário a implementação por parte do usuário. O Processing possui a função “noise()” que aceita até 3 argumentos, gerando assim um ruído de uma, duas ou 3 dimensões. (PROCESSING, 2022)

No Unity (UNITY, 2022) existe a função “Mathf.PerlinNoise(float x, float y)”. Essa função gera apenas ruídos em duas dimensões. Não existe uma função do próprio Unity para geração do ruído em três dimensões, mas é possível implementar soluções para isso.

3.1.1 Implementação do ruído

O ruido foi implementado utilizando o *Unity* para que o resultado pudesse ser observado, e o código foi escrito na linguagem C# através do editor *Visual Studio Code*.

Para utilizar a função de ruído, deve-se informar os valores x e y. Esses valores vão gerar o ruído naquela coordenada específica do mapa.

O valor de x é obtido através da coordenada de onde se está no mapa divido pela escala (zoom da Imagem), multiplicado pela frequência e somado com o *offset* (deslocamento de cada uma das coordenadas no eixo x ou y).

$$x = \left(\frac{Coordx}{Escala}\right) * frequencia + offsetX$$

O valor de Y é gerado de maneira semelhante, mas a multiplicação é feita pela amplitude e não pela frequência.

$$y = \left(\frac{CoordY}{Escala}\right) * Amplitude + offsetY$$

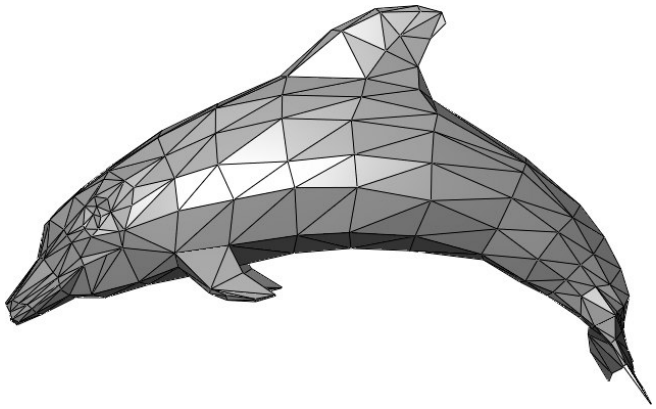
Após a obtenção dos valores de x e y, seus valores serão utilizados para gerar o ruido utilizando a função de *perlin noise* do *Unity*.

3.2 Textura

Após a obtenção do mapa de ruído atribui-se cores aos valores. Por exemplo, caso o valor do ruído no ponto seja menor que 0,4, aquele ponto será pintado de azul, caso seja maior que 0,9 será vermelho. A partir daí pode-se utilizar *meshes* para começar a desenvolver um mapa em 3D.

Na computação gráfica, *Mesh* é o conjunto de polígonos utilizados para dar forma aos objetos 3D. A Figura 9 demonstra a modelagem do *mesh* de um golfinho.

Figura 9: Exemplo de um *mesh* triangular representando um golfinho.



Fonte: (WIKIPEDIA, [20--])

É possível observar na Figura 9 que a imagem é formada de vários triângulos, quadriláteros e outros polígonos para formar a imagem de um golfinho. Essa representação tem uma quantidade relativamente baixa de polígonos. Quando se deseja ter uma imagem mais realista é necessário a utilização de mais polígonos.

3.3 Ferramentas utilizadas.

As ferramentas que foram escolhidas para serem utilizadas neste trabalho foram o *Unity*, um motor gráfico com várias ferramentas de fácil acesso para utilização e capaz da geração de gráficos 2D e 3D. E o *Visual Studio Code*, um editor e interpretador de código. O Visual Studio Code foi escolhido pois ele reconhece as funções de funcionamento *Unity*, facilitando assim o desenvolvimento quando ambos são usados em conjunto.

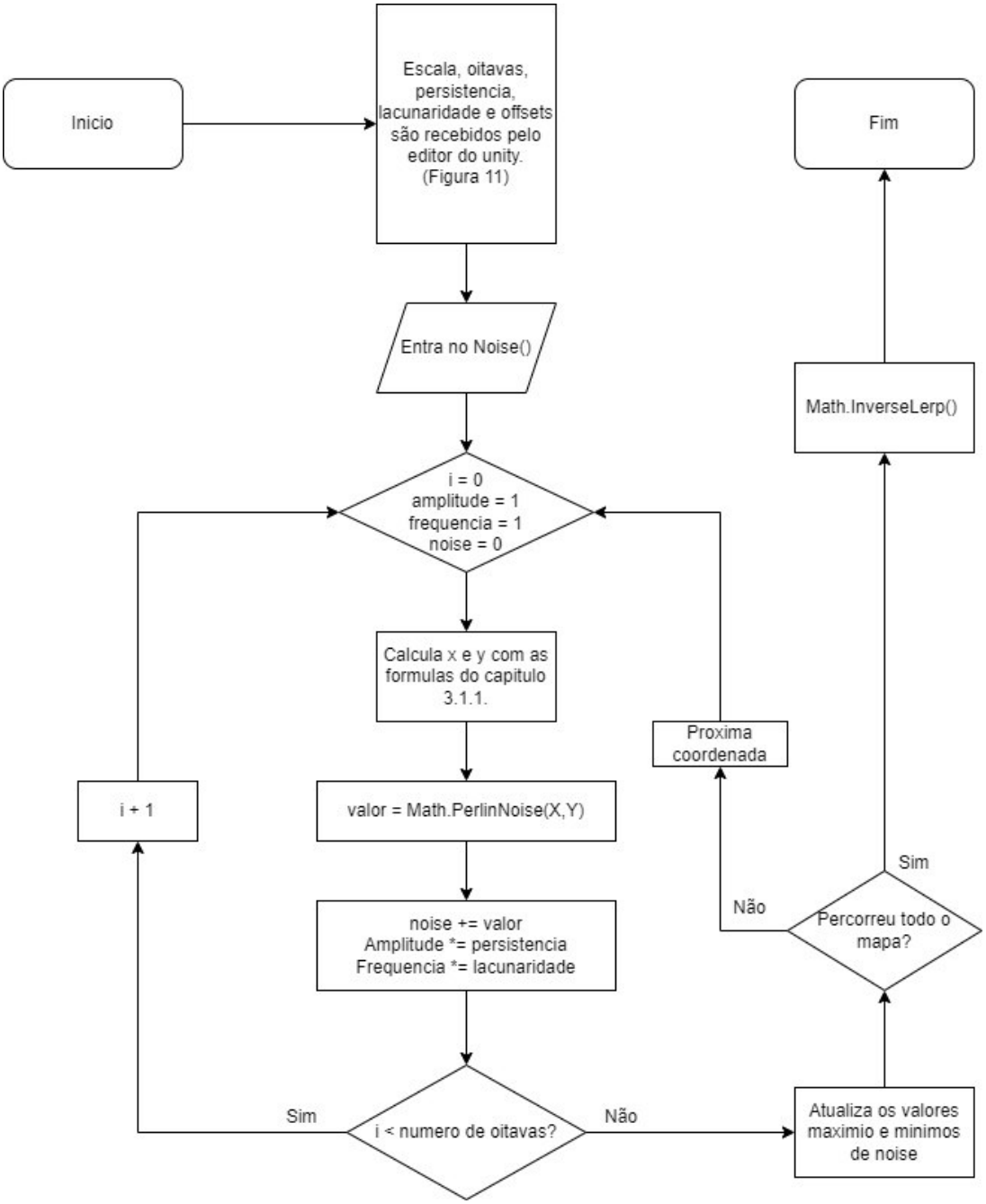
4 DESENVOLVIMENTO

Neste Capítulo será mostrado o código desenvolvido neste trabalho.

4.1 Ruído

O algoritmo deste trabalho foi elaborado seguindo uma sequência de passos. Esses passos podem são ilustrados no fluxograma da Figura 10. Logo em seguida temos a explicação dos passos.

Figura 10: Fluxograma do método noise.cs desenvolvidos pelo autor.



Fonte: Autor.

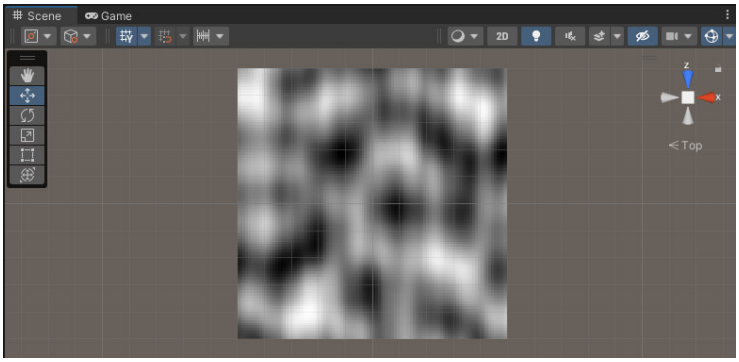
Existem dois loops de repetição para que o algoritmo seja executado em cada coordenada do mapa. Antes de se entrar no loop que processará as oitavas a amplitude, a frequência e a variável "*noise*", que servirá para guardar o valor total do Ruído durante a execução de cada oitava, são inicializadas com os valores de 1, 1 e 0 respectivamente.

Dentro do loop de repetição as coordenadas x e y serão processadas em relação a escala, frequência/amplitude e offsets. Após as oitavas é verificado se é necessário atualizar os valores máximos e mínimos do Ruído e o *noise* é salvo naquela coordenada do mapa de Ruído.

Por último entramos em outro laço de repetição. Nele os valores do mapa de Ruído serão interpolados através da função "*inverseLerp*". A função indica o quão próximo um valor está entre os valores indicados. O retorno está entre 0 e 1. Se o retorno for 0 ele é igual ao valor mínimo, se for 0.5 ele está no meio entre o valor máximo e mínimo, e assim por diante. A função tem como entrada o valor mínimo do Ruído, o valor máximo do Ruído e o ponto em que se deseja calcular. Após isso é retornado o *noiseMap*.

Na Figura 11 é mostrado o resultado da geração do ruido. O resultado é um mapa de ruido e é a partir dele que o mapa do jogo será feito.

Figura 11: Mapa de ruído utilizado na geração do mapa na Figura 13.



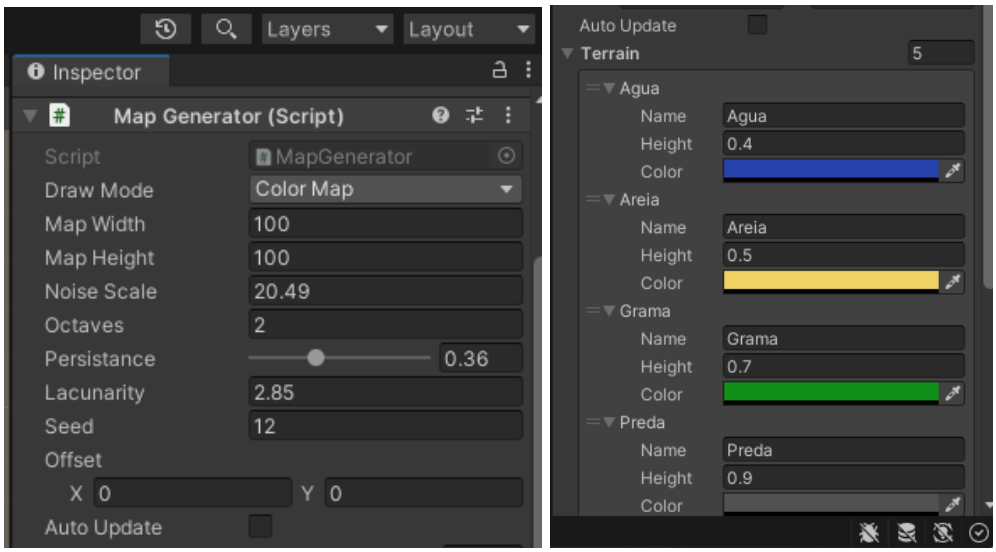
Fonte: Autor

4.2 Mapa

A partir do editor do *Unity* é possível inserir os valores que serão usados na geração do ruído e dos terrenos. Os terrenos são criados atribuindo uma cor e um nome a cada um dos pontos gerados no mapa de ruído. No caso mostrado na Figura 12, caso o valor seja igual ou inferior a 0,4 ele será pintado de azul e servirá como

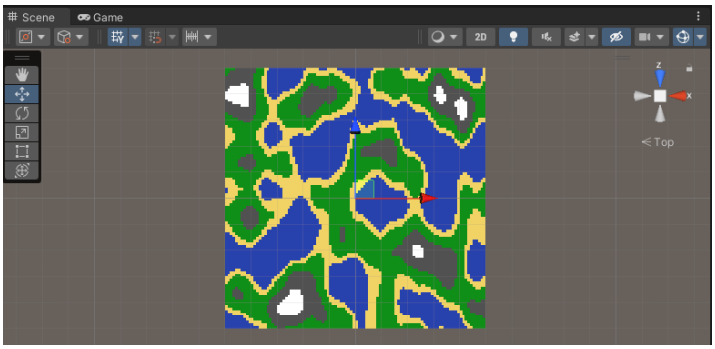
água. Se estiver entre 0,5 e 0,4 ele será pintado de amarelo e receberá o nome de areia. Se estiver entre 0,7 e 0,5 será pintado de verde e receberá o nome de grama. Se estiver entre 0,9 e 0,7 será pintado de cinza e receberá o nome de pedra. Se estiver acima de 0,9 será pintado de branco e receberá o nome de neve. Na Figura 13 está o resultado da geração com valores mostrados na Figura 12.

Figura 12: Os editores de mapa feitos no *Unity*.



Fonte: Autor

Figura 13: Resultado da coloração do mapa de Ruído.



Fonte: Autor

4.3 Utilização

Os mapas gerados são apenas ferramentas para o que se deseja desenvolver. No caso do mapa mostrado na Figura 13, ele poderia ser usado como um nível em jogos de estratégia como *Final Fantasy tatics* ou *Fire Emblem.* A Figura 14 mostra um exemplo de level no jogo *Fire Emblem Blazing Sword*. O mapa poderia se adequar

para um jogo com a jogabilidade semelhante a esse já que o ambiente afeta as batalhas. Um arqueiro teria vantagem atacando em terreno mais alto ou apenas tropas que voam poderiam passar por terrenos com água por exemplo.

Figura 14: Exemplo de level no jogo *fire emblem Blazing sword*.

Fonte: (PIETRIOTS, 2016)

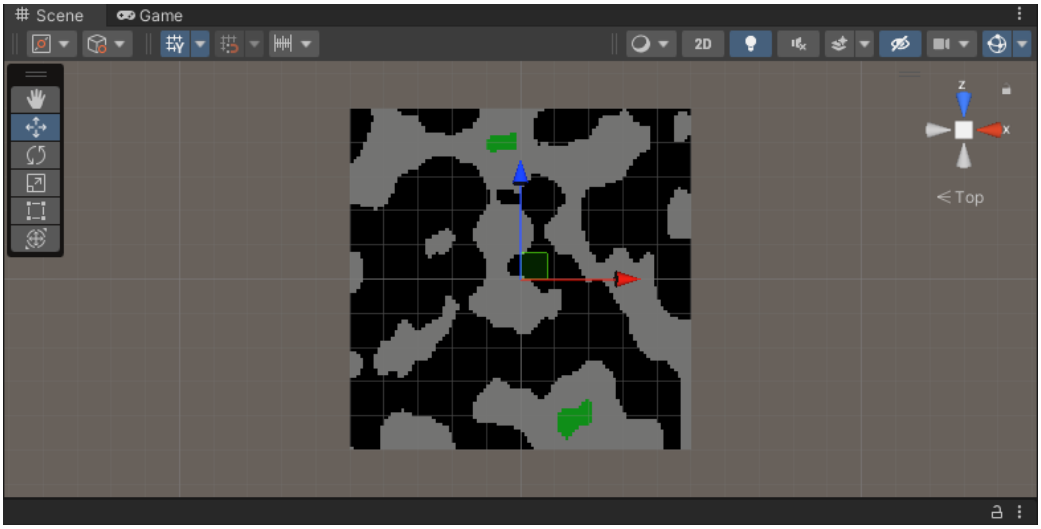
Com algumas mudanças no editor, o mapa pode ser mudado para um estilo diferente. Foi feita uma leve diminuição no valor da persistência e um aumento no valor da lacunaridade. Além disso os valores e nomes de terreno foram alterados. Na Figura 15 um exemplo de mapa que pode ser utilizado em um jogo estilo *Roguelike.* O jogo que deu nome a esse estilo se chama *rogue*. Nele o jogador controla um aventureiro e deve entrar em masmorras para pegar tesouros. Cada quarto é gerado aleatoriamente e ligado por corredores.

Figura 15: Exemplo de level do jogo *Rogue*.

Fonte (WIKIPEDIA, [20--])

Para funcionar com esse estilo de jogo os terrenos foram mudados. Foi feita uma leve diminuição no valor da persistência e um aumento no valor da lacunaridade. Além disso os valores e nomes de terreno foram alterados. Em cinza são os locais acessíveis, o terreno onde o jogador pode se movimentar e interagir. Em preto as delimitações do mapa, onde o jogador não pode acessar. Os pontos verdes são locais importantes, onde o jogador encontrará itens ou colecionáveis.

Figura 16: Mapa com alterações para se adequar ao estilo *roguelike*.



Fonte: Autor

5 RESULTADOS OBTIDOS

O desenvolvimento deste trabalho resultou na criação de um mapa. O mapa é simples e foi criado com o objetivo de ser mais genérico para que possa ser aplicado a gêneros de jogos diferentes com pouco esforço. Apesar disso seria necessário que mais esforço para que os resultados pudessem ser utilizados de maneira mais eficaz em um jogo.

Dependendo do estilo de jogo no qual o mapa será utilizado as mudanças deveriam ser diferentes. No caso do exemplo de um jogo roguelike talvez seja necessário utilizar outro tipo de ruido para que os caminhos gerados sejam menos como ilhas e mais como longos caminhos conectados para facilitar a navegação do jogador e diminuir a chance de que ele fique preso em um local sem saída. Usando como exemplo a Figura 16, caso o jogador tivesse como ponto inicial uma das ilhas isoladas ele não teria como progredir e teria que reiniciar o jogo.

No caso do exemplo do jogo de estratégia será necessário a adição de outros elementos para criar diversidade de jogo e aumentar a complexidade do jogo. Da maneira como foi apresentado na Figura 13 o ambiente é o único elemento que poderia ser variado para criar desafio de jogo. A inserção de elementos como cidades ou florestas densas irão aumentar a variedade de possibilidades que podem ser usadas pelo jogador. Para a inserção desses elementos poderia ser utilizado outro método de ruido que os espalhariam em locais adequados pelo mapa.

A criação dos *meshes* no mapa é outro elemento que aumentaria o numero de possibilidades de uso, pois o mapa poderia deixar de ser usado apenas em 2d com uma visão superior para ser usado também em 3d.

6 CONSIDERAÇÕES FINAIS

Este trabalho teve como objetivo desenvolver um método que utilize PCG para criar um mapa. Para isso, foi demonstrado algoritmos que geram ruido e como o ruido pode ser utilizado para gerar um mapa.

Perlin noise, wavelet noise e spot noise foram apresentados para exemplificar diferentes métodos de geração de ruido. O *Perlin noise* é um dos ruídos mais famosos e utilizados. O *wavelet noise* é uma melhora do ruido de perlin que aumenta a sua distância de renderização. O *spot noise* é mais utilizado para a visualização de dados do que para a computação gráfica. Desses o *Perlin noise* foi o escolhido para ser utilizado no desenvolvimento do trabalho devido a sua popularidade e a facilidade em que poderia ser gerado.

exemplos de jogos que utilizam de PCG foram apresentados, e foi citado sobre como *Minecraft* utiliza da geração procedural para gerar seus biomas, terrenos, vegetação e cavernas.

Foi mostrado o resultado do algoritmo desenvolvido e foi dado exemplos de como poderia ser utilizado em dois estilos de jogos diferentes. Também foi explicado maneiras de como seria possível melhorar o algoritmo de maneira geral e dependendo do estilo de jogo em que o mapa será utilizado.

6.1 Trabalhos futuros

Em trabalhos futuros é recomendado que se tome as seguintes medidas:

- Desenvolvimento de outros métodos de ruido para geração de mais elementos visuais ou de pontos importantes.
- Aplicação de *meshes* para que o mapa possa ser visualizado e/ou utilizado em 3D.

REFERENCIAS BIBLIOGRAFICAS

BALDWIN, A. et al. Mixed-Initiative Procedural Generation of Dungeons using Game Design Patterns. **IEEE Conference on Computational Intelligence and Games**, 2017.

CEFALA ORG. **o som**. [S.l.]. 2008.

COMPTON, K. Practical Procedural Generation for Everyone, 31 Maio 2017. Disponivel em: <https://www.youtube.com/watch?v=WumyfLEa6bU>. Acesso em: 29 Novembro 2022.

COOK, R. L.; DEROSE, T. **Wavelet Noise**. Pixar Animation Studios. [S.l.], p. 9. 2005.

DIGIDIGGER. How does procedural generation work? | Bitwise, 29 Janeiro 2020. Disponivel em: <https://www.youtube.com/watch?v=-POwgollFeY>. Acesso em: 29 Novembro 2022.

DUARTE, P. M. Geração Procedural de Cenários Orientada a Objetivos, Natal, 2012.

FONETICA E FONOLOGIA. **O som**. UFMG. [S.l.]. 2008. Disponivel em: <https://www.cefala.org/fonologia/acustica_osom_2.php>. Acesso em: 07 Dezembro 2022

DUNGEON GENERATION IN BINDING OF ISAAC, **Boris**, 12 Setembro 2020. Disponível em: <https://www.boristhebrave.com/2020/09/12/dungeon-generation-in-binding-of-isaac/>. Acesso em: 06 Dezembro 2020.

FIRE EMBLEM (GBA) – INITIATION, **Pietriots**, 05 Maio 2016, Disponivel em: < https://pietriots.com/2016/05/05/fire-emblem-gba-initiation/>. Acesso em: 07 Dezembro 2022.

GALIN, E. et al. Procedural Generation of Roads, 2010.

GOD OF WAR RAGNAROK BUGET AND COSTS, **What box game**, 1 Setembro 2022**.** Disponível em: <https://whattheboxgame.com/god-of-war-ragnarok-budget-and-costs/>. Acesso em: 06 Dezembro 2022.

HOTZ, I. et al. **Tensor-Fields Visualization Using a Fabric-like Texture Applied to Arbitrary Two-dimensional Surfaces**. Institute id Data Analisys and Visualization (IDAV). California. 2009.

LAGAE, A. et al. **Procedural Noise using Sparse Gabor Convolution**. ACM. [S.l.]. 2009.

LAGAE, A. et al. **A Survey of Procedural Noise Functions**. [S.l.]. 2010.

LEITE, G. D. O. B.; LIMA, E. S. D. Geração Procedural de Mapas para Jogos 2D, Teresina, 11 - 13 Novembro 2015.

LEVEL GENERATION, **The binding of Isaac Rebirth wiki**, Sem data, Disponivel em: <https://bindingofisaacrebirth.fandom.com/wiki/Level_Generation>. Acesso em: 06 Dezembro 2022.

LINDEN, R. V. D.; LOPES, R.; BIDARRA, R. Procedural Generation of Dungeons. **TRANSACTIONS ON COMPUTATIONAL INTELLIGENCE AND AI IN GAMES**, março 2014.

PERLIN NOISE, **Wikipedia**, sem data, Disponível em: < https://en.wikipedia.org/wiki/Perlin_noise>. Acesso: 07 Dezembro 2022.

PROCESSING. **Processing Foundation**, 2022. Página inicial. Disponível em: <https://processing.org/>. Acesso em: 06 Dezembro 2020.

ROGUE (VIDEO GAME), **Wikipedia**, sem data, Disponivel em: < https://en.wikipedia.org/wiki/Rogue_%28video_game%29>. Acesso em: 07 Dezembro 2022.

UNITY. **Unity Technologies**, 2022. Página inicial. Disponível em: <https://unity.com/pt>. Acesso em: 06 Dezembro 2020.

WIJK, J. J. V. **Spot Noise Texture Synthesis for Data Visualization**. SIGGRAPH 91. [S.l.]: [s.n.]. 1991.

PONTIFÍCIA UNIVERSIDADE CATÓLICA DE GOIÁS
GABINETE DO REITOR

Av. Universitária, 1069 • Setor Universitário
Caixa Postal 86 • CEP 74605-010
Goiânia • Goiás • Brasil
Fone: (62) 3946.1000
www.pucgoias.edu.br • reitoria@pucgoias.edu.br

RESOLUÇÃO n° 038/2020 – CEPE

ANEXO I

APÊNDICE ao TCC

Termo de autorização de publicação de produção acadêmica

O(A) estudante **Guilherme Neves Canêdo** do Curso de **Ciências da Computação**, matrícula **2016.2.0028.0240-1**, telefone: **(62) 99640-4414** e-mail **guinc159@gmail.com**, na qualidade de titular dos direitos autorais, em consonância com a Lei nº 9.610/98 (Lei dos Direitos do Autor), autoriza a Pontifícia Universidade Católica de Goiás (PUC Goiás) a disponibilizar o Trabalho de Conclusão de Curso intitulado **Geração de mapas utilizando geração procedural**, gratuitamente, sem ressarcimento dos direitos autorais, por 5 (cinco) anos, conforme permissões do documento, em meio eletrônico, na rede mundial de computadores, no formato especificado (Texto(PDF); Imagem (GIF ou JPEG); Som (WAVE, MPEG, AIFF, SND); Vídeo (MPEG, MWV, AVI, QT); outros, específicos da área; para fins de leitura e/ou impressão pela internet, a título de divulgação da produção científica gerada nos cursos de graduação da PUC Goiás.

Goiânia, 10 de dezembro de 2022.

Assinatura do autor: Guilherme Neves Canedo

Nome completo do autor: Guilherme Neves Canêdo

Assinatura do professor – orientador: Gustavo Siqueira Vinhal

Nome completo do professor – orientador: Gustavo Siqueira Vinhal