UNIVERSIDADE FEDERAL DA FRONTEIRA SUL
CAMPUS DE CHAPECÓ
CURSO DE CIÊNCIA DA COMPUTAÇÃO

ESTELA MARIS VILAS BOAS

CONTRIBUIÇÕES PARA GERAÇÃO PROCEDURAL DE TERRENOS

UM ESTUDO SOBRE AGENTES INTELIGENTES E STEERING BEHAVIORS

CHAPECÓ
2021

ESTELA MARIS VILAS BOAS

CONTRIBUIÇÕES PARA GERAÇÃO PROCEDURAL DE TERRENOS

UM ESTUDO SOBRE AGENTES INTELIGENTES E STEERING BEHAVIORS

Trabalho de conclusão de curso apresentado como requisito para obtenção do grau de Bacharel em Ciência da Computação da Universidade Federal da Fronteira Sul.
Orientador: Prof. Dr. Fernando Bevilacqua

CHAPECÓ

2021

Vilas Boas, Estela Maris

Contribuições para geração procedural de terrenos: um estudo sobre agentes inteligentes e steering behaviors / Estela Maris Vilas Boas. – 2021.

52 f.: il.

Orientador: Prof. Dr. Fernando Bevilacqua.

Trabalho de conclusão de curso (graduação) – Universidade Federal da Fronteira Sul, curso de Ciência da Computação, Chapecó, SC, 2021.

1. Geração Procedural. 2. Agentes Inteligentes. 3. Steering Behaviors. I. Bevilacqua, Prof. Dr. Fernando, orientador. II. Universidade Federal da Fronteira Sul. III. Título.

E-mail: estelavilasboas01@gmail.com

ESTELA MARIS VILAS BOAS

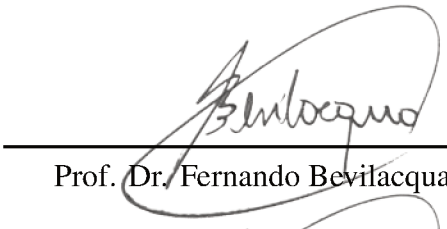
CONTRIBUIÇÕES PARA GERAÇÃO PROCEDURAL DE TERRENOS
UM ESTUDO SOBRE AGENTES INTELIGENTES E STEERING BEHAVIORS

Trabalho de conclusão de curso apresentado como requisito para obtenção do grau de Bacharel em Ciência da Computação da Universidade Federal da Fronteira Sul.

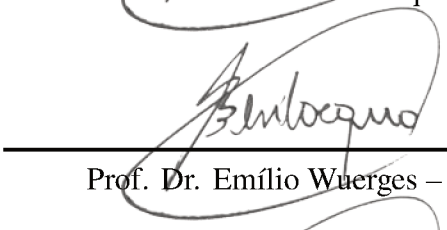
Orientador: Prof. Dr. Fernando Bevilacqua

Este trabalho de conclusão de curso foi defendido e aprovado pela banca avaliadora em: 18/10/2021

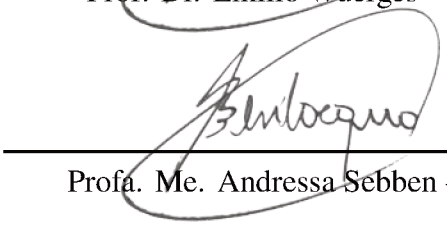
BANCA AVALIADORA



Prof. Dr. Fernando Bevilacqua – UFFS



Prof. Dr. Emílio Wuerges – UFFS



Profa. Me. Andressa Sebben – UFFS

AGRADECIMENTOS

Quando comecei a escrever esse trabalho, tinha uma ideia completamente diferente do que escreveria nos agradecimentos. Tornei-me alguém diferente com o passar dos meses, com ideias e questionamentos novos. Concluir esse trabalho é o fim de uma das muitas jornadas que ainda estão por vir. Sendo assim, não acho justo escrever e agradecer como na minha ideia inicial.

Posso começar agradecendo as contribuições do orientador Fernando Bevilacqua, que fez muitos comentários motivadores ao longo do processo e me manteve animada para continuar escrevendo. Posso agradecer as sugestões dos membros da banca Emílio Wuerges e Andressa Sebben, que influenciaram muito no desenvolvimento desse trabalho com seus comentários escritos ou falados sempre em minha mente, por mais que eu, às vezes, tenha decidido não seguir algumas das sugestões.

Ainda tenho que agradecer meus pais, Maristella e Carlos Vilas Boas. Eles me incentivaram a finalizar este trabalho o mais rápido que eu conseguisse, sempre me perguntando como as coisas andavam e me motivando de formas bem inusitadas.

Por toda a paciência aos muitos surtos e por todas as sugestões de como melhorar este trabalho, preciso agradecer o meu futuro noivo, Andrew Malta. E digo "futuro noivo", pois eu ainda tenho que aceitar um dos muitos pedidos de casamento que ele já me fez. Quem sabe um dia.

E por último, vou escrever a minha ideia inicial. Quero agradecer aquela garotinha que chorava debaixo das cobertas com medo, que olhava sonhadora para o futuro. Se ela não tivesse sido corajosa e persistente, eu não teria me tornado quem sou hoje. Talvez seja estranho pensar assim, mas este trabalho é também de certa forma uma dedicatória para aquela pequena escritora de fantasias.

“Don’t think. Become!”

(Guillermo Del Toro, Trollhunters)

RESUMO

A indústria e o mercado dos jogos digitais estão em constante crescimento, com jogadores cada vez mais exigentes e demandando mais conteúdo. A criação desse conteúdo requer investimento de tempo e dinheiro que, se não bem administrado, pode trazer prejuízo à produção de outros elementos do jogo. Com isso, a geração procedural se apresenta como uma possível solução para criação de conteúdo sem consumir o tempo de trabalho de uma equipe e mantendo a qualidade no produto final. Este trabalho apresenta a proposta de uma análise a viabilidade da aplicação de conceitos de geração procedural, focado na criação de terrenos, aplicando agentes de software com *steering behaviors*. A execução dos experimentos apresentou possibilidade de controle sobre a geração de um mapa de alturas de um terreno de acordo com os *steering behaviors* e parâmetros utilizados.

Palavras-chave: Geração Procedural. Agentes Inteligentes. Steering Behaviors.

ABSTRACT

The video games industry and market show constant growth, and players are demanding more content. Create this content requires time and money investment that, if not well managed, can damage the production of other game elements. Thus, procedural generation presents itself as a solution for creating content without consuming a team's time and maintaining quality in the final product. This work presents the proposal of analysis about the viability of the application of procedural generation concepts, focused on the creation of terrain, applying software agents with steering behaviors. The execution of experiments presented the possibility of controlling the generation of a terrain heightmap according to the chosen steering behaviors and used parameters.

Keywords: Procedural Generation. Intelligent Agents. Steering Behaviors.

LISTA DE ILUSTRAÇÕES

SUMÁRIO

1 INTRODUÇÃO

Jogos digitais são elementos pervasivos na sociedade contemporânea, sendo eles consumidos como forma de diversão e entretenimento. Disponíveis nos mais diversos formatos, como jogos para computadores, consoles e dispositivos móveis, centenas de milhares de pessoas são entretidas por jogos digitais diariamente (11). Possuindo um expressivo número de propostas e estilos de jogabilidade diferentes, cada jogo digital possui suas próprias características e atingem diferentes tipos de pessoas.

Independentemente da temática ou mecânica do jogo, seu conteúdo é um elemento significativamente importante na experiência dos jogadores. Em um jogo com temática musical, por exemplo, jogadores esperam elementos musicais que atendam seus anseios em termos de qualidade e entretenimento. Similarmente, em um jogo que possibilite ao jogador explorar grandes áreas de terreno, a paisagem e sua diversidade de conteúdo são fatores chave para uma experiência satisfatória.

Considerando essa diversidade de conteúdo, o mercado de jogos está em constante crescimento e novos títulos são lançados e disponibilizados a um público que está cada vez mais exigente. Visto que jogos são produtos multidisciplinares que envolvem esforços de diferentes áreas e profissionais, como música, roteiro, arte e software, a escolha de investir na produção de conteúdo pode causar detrimento em diversos elementos de um jogo, além de desgastar os profissionais que nele trabalham.

Por um lado, o produto final (o jogo) precisa ter um conteúdo de qualidade para que jogadores se interessem em investir seu tempo jogando-o ou, no mínimo, em comprá-lo. Por outro lado, a demanda dos jogadores por mais e melhores conteúdos impacta diretamente na complexidade de produção do jogo (8), o que invariavelmente impacta em seu custo. Encontrar o equilíbrio entre qualidade e quantidade do conteúdo com os custos envolvidos pode ser uma tarefa muito complexa.

Portanto, estúdios são forçados a investir cada vez mais recursos na produção de conteúdo para seus jogos, o que encarece o produto final e diminui a margem de lucro. Além disso, o investimento financeiro é limitado e por vezes escasso para estúdios desenvolvedores de jogos, o que torna a matemática de produção ainda mais complexa. Dependendo da situação, até o tempo disponível para a produção pode se tornar um recurso escasso, podendo culminar no desgaste dos profissionais envolvidos na produção.

Nesse contexto, a geração procedural de conteúdo (GPC) apresenta-se como uma solução viável. Um conteúdo gerado proceduralmente é, em linhas gerais, algo produzido como resultado de um processo descrito através de uma representação matemática. Um dos grandes benefícios da GPC é sua flexibilidade, pois é possível utilizá-la para a geração dos mais diversos tipos de conteúdos num curto espaço de tempo e com o mínimo de intervenção.

Possibilitando desde a geração de cidades e estradas à geração plantas e terrenos, a GPC pode ser uma grande aliada no desenvolvimento de um jogo digital. Se assim utilizada, é

possível reduzir consideravelmente os custos de produção ao gerar uma ampla gama de conteúdo procedural. Por exemplo, jogos de mundo aberto geralmente disponibilizam ambientes naturais de grande escala para que jogadores possam explorar (9). Produzir esses vastos ambientes requer muitos recursos, a não ser que GPC seja utilizada.

Embora o conteúdo dessas áreas demande um certo nível de qualidade, como já mencionado, dificilmente jogadores explorarão todo o terreno disponível. Há uma certa tendência de permanência nos locais chave do enredo do título, como cidades, vilas e pontos principais. Consequentemente, locais que não são chaves poderiam ser gerados a partir de algoritmos de GPC a fim de economizar recursos e gerar conteúdo sem muito prejuízo à qualidade final do jogo.

A utilização de GPC no contexto da produção de jogos digitais é um tópico amplamente mencionado na literatura (7). Mesmo que parte do conteúdo do jogo seja gerado através de uma técnica de GPC, ele ainda precisa ser contextualizado na temática do jogo em questão. Por exemplo, terrenos criados com técnicas de GPC para um jogo de faroeste idealmente devem evitar produzir áreas tropicais contendo coqueiros, praias e outros elementos tropicais.

Portanto, é conveniente que elementos criados por GPC sejam parametrizáveis e controláveis pela equipe que está produzindo este conteúdo. Isso garante que o conteúdo gerado esteja alinhado com as muitas características empregadas nas demais partes do jogo, como temática, mecânica, direção de arte e muitas outras. Deixar o conteúdo à mercê apenas dos procedimentos matemáticos e da aleatoriedade pode gerar inconsistências indesejadas.

Portanto, De Carli et al. (7) definem duas grandes categorias de técnicas de GPC aplicadas ao desenvolvimento de jogos: assistidas e não-assistidas. Uma técnica assistida exige que uma pessoa produza um material inicial que será utilizado como base pela técnica para geração de mais conteúdo, enquanto uma não-assistida é capaz de gerar conteúdo sem necessidade de intervenção de um operador.

Ambas as técnicas possuem determinados parâmetros e regras que podem ajudar a controlar os conteúdos gerados. Todavia, quanto maior a necessidade de interação do operador, mais dependente a técnica se torna, o que reduz sua utilidade no ponto de vista do investimento de tempo. Se a técnica demanda muito tempo de trabalho para parametrização e ajuste de qualidade por conta de sua natureza aleatória intrínseca, pode-se incorrer o risco de que a produção deste conteúdo seja mais vantajosa sem a utilização de GPC.

Considerando este aspecto, a técnica de GPC ideal seria uma técnica não-assistida capaz de gerar uma grande quantidade de conteúdo de forma que os parâmetros possam ser facilmente ajustados para garantir a contextualização de todos os elementos do jogo. A produção de técnicas com essas características tem sido um tema muito recorrente de pesquisa nessa área. Garantir tais aspectos é uma tarefa complexa que torna a produção de técnicas de GPC não-assistidas um desafio, pois a controlabilidade é um grande fator de dificuldade.

Em luz ao que fora descrito, nota-se a importância da realização de pesquisas em técnicas de GPC não-assistida de fácil parametrização por parte dos operadores. Estas técnicas

dependem da aleatoriedade para gerar uma boa diversidade de conteúdos, mas devem apresentar boa controlabilidade. Possibilitar que o operador entenda as consequências de cada parâmetro é importante, pois favorece a flexibilidade do algoritmo e permite customização e ajustes sem sacrificar a variabilidade aleatória.

Uma forma de obter essas características é utilizando agentes de software nas técnicas de GPC. Agentes são elementos que seguem regras de negócio simples, mas cuja interação entre si resulta em comportamentos, simulações e conteúdos mais complexos. A utilização de agentes em GPC não é um tópico inédito, especialmente na geração de terrenos, e as técnicas discutidas apresentam as qualidades mencionadas.

Agentes podem ser parametrizados de forma distinta, o que impacta em diferentes níveis de interação e resulta em conteúdos variados e parametrizáveis. Um tipo de técnica de movimentação de agentes de software, empregado principalmente em jogos digitais, são *steering behaviors* (19). Originalmente proposta por Craig W. Reynolds, *steering behaviors* [1] (ou SB) foram criados para modelar movimentação de atores em grupo, como revoadas de pássaros ou debandada de animais, mas acabaram por ser empregados em outros contextos, como na geração procedural de terrenos.

As técnicas *steering behaviors* são categorizadas em diferentes tipos, como *seek*, *pursuit* e *avoidance* [2]. Independente da categoria, *steering behaviors* utilizam informações locais de cada agente para calcularem suas interações. Como resultado, ocorre a simulação de movimentações complexas com uma maior riqueza de detalhes, porém sem um investimento proporcional de parametrização de um operador.

Não há uma extensa investigação na literatura da geração de conteúdos para jogos digitais, particularmente terrenos, utilizando-se técnicas de GPC baseadas em agentes. Especificamente, há uma falta de informações na utilização de GPC baseada em agentes guiados por *steering behaviors*. Portanto, esse trabalho apresenta uma proposta de investigação sobre a criação de mapas de altura para geração de terrenos através do uso de agentes e *steering behaviors*.

O objetivo principal foi testar a viabilidade do emprego de *steering behaviors* para parametrizar a geração procedural de terrenos de terrenos, almejando a criação de uma técnica não assistida. Conforme mencionado, o balanço ideal entre parametrização e geração de conteúdo é o foco de uma técnica não assistida. Logo, através desse trabalho e de seus resultados, espera-se contribuir com um melhor entendimento sobre os impactos que a controlabilidade pode causar na geração de terrenos ao empregar agentes baseados em *steering behaviors*.

Para concretizar tais objetivos, os seis capítulos a seguir visam melhor descrever o estudo que foi realizado. O Capítulo 2 melhor especifica cada objetivo visado por este trabalho. Por sua vez, o Capítulo 3 apresenta alguns argumentos que explicam a importância dos objetivos definidos. No Capítulo 4 são apresentados os principais conceitos necessários para o desenvol-

[1] *Steering behaviors* podem ser traduzidos para o português como *comportamentos de direção*.

[2] Originalmente, os nomes das categorias de *steering behaviors* são utilizados em Inglês. Nesse caso, as categorias seriam *busca*, *perseguição* e *rejeição* na tradução para o Português, mas, para fins de clareza e consistência com a literatura, os termos serão mantidos no idioma original.

vimento deste trabalho. O Capítulo 5 apresenta alguns trabalhos que lidaram com este tema. Finalmente, o Capítulo 6 apresenta a maneira na qual este trabalho empregou as ferramentas disponíveis para concretização de seus objetivos, que será seguido do Capítulo 7, onde são apresentados os resultados dos experimentos.

2 OBJETIVOS

2.1 OBJETIVOS GERAIS

Considerando o contexto introduzido, os objetivos gerais deste trabalho consistem em investigar a utilização de agentes de software baseados em *steering behaviors* na construção de um mapa de alturas para geração procedural de terrenos, os quais são empregados como conteúdos para jogos digitais.

2.2 OBJETIVOS ESPECÍFICOS

- Delinear o estado da arte em relação aos conceitos de GPC, agentes de software e *steering behaviours*;
- Estabelecer a viabilidade do emprego de *steering behaviors* para parametrizar um mapa de alturas, uma técnica de geração procedural de terrenos;
- Analisar, no contexto da qualidade e corretude, o conteúdo procedural relacionado a terrenos produzido por agentes guiados por *steering behaviors*;
- Desenvolver testes e investigação do emprego de agentes de software parametrizáveis em geração procedural de conteúdos.

3 JUSTIFICATIVA

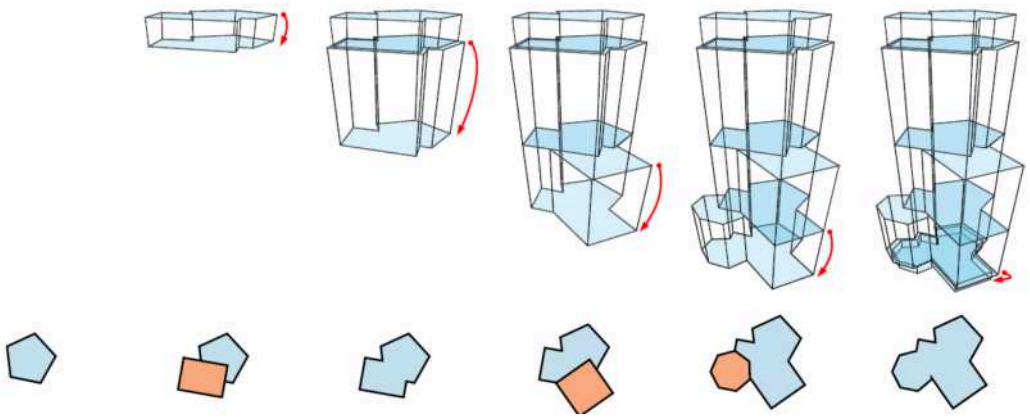
Fundamentadas em um conjunto de parâmetros, limitadores e regras, técnicas de GPC são baseadas na aleatoriedade das iterações. Algumas dessas técnicas podem ter uma capacidade de geração de conteúdo infinita, como é o caso das técnicas baseadas em fractais. Um fractal é uma função matemática recursiva que é capaz de, no contexto de GPC, produzir detalhes infinitos, pois novas informações são produzidas e adicionadas a cada iteração (14).

Suponha que uma equipe precisa produzir o conteúdo de uma cidade completa para um jogo. Esse conteúdo pode ser modelado e texturizado peça por peça, como habitações, ruas, calçadas, etc. Tal tarefa demandaria um esforço considerável em horas de trabalho, principalmente se for uma grande cidade, impactando diretamente o custo de produção. No entanto, se essa equipe utilizar uma técnica de GPC baseada em fractais, por exemplo, a diversidade dos resultados será significativa e demandará menos tempo para ser obtida. Dependendo dos parâmetros e do número de iterações realizadas, as possibilidades são cada vez maiores.

No entanto, embora produzir conteúdo em abundância seja importante, isso não é necessariamente desejável quando se trata de fractais. Às vezes, funções fractais podem gerar detalhes demais e talvez até mesmo irreais. Por exemplo, habitações e prédios com muitos cantos ou uma variação demasiada na estrutura de um andar para o outro não são características desejáveis na maioria dos casos. Portanto, a equipe encarregada dessa tarefa deve possuir ferramentas que possibilitem uma boa controlabilidade da geração deste conteúdo a fim de garantir sua qualidade.

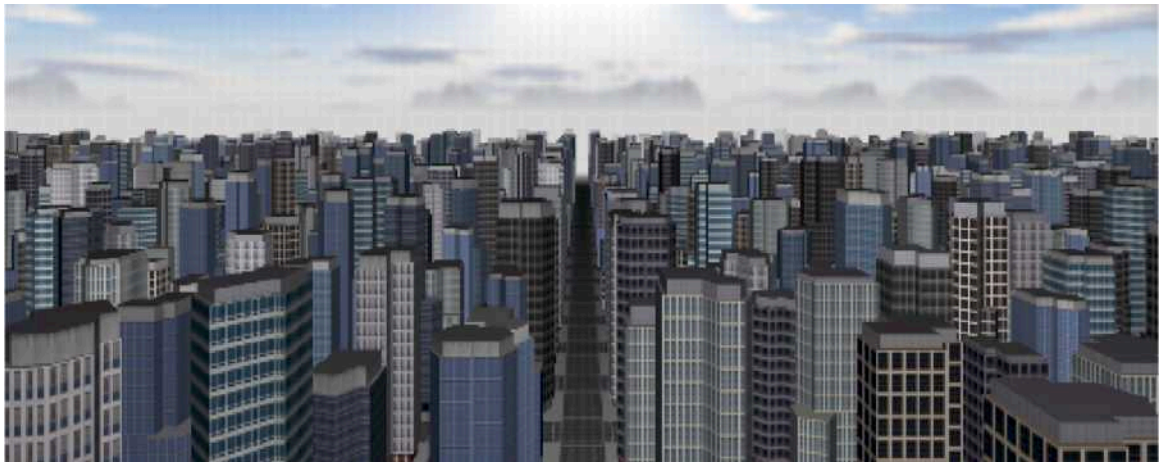
Uma das formas de controle que a equipe teria é a limitação na aleatoriedade do método durante a geração das habitações. O processo, conforme ilustrado na Figura 1, mostra o desdobramento de uma técnica de GPC aplicada a geração de prédios em uma cidade. Escolhendo-se o conjunto de formas geométricas que o algoritmo utilizará e os parâmetros que limitam a aleatoriedade de escolha dessas formas, pode-se obter como resultado a geração de diferentes prédios, conforme apresentado na Figura 2.

Figura 1 – Desdobramento dos passos de uma técnica de GPC para produção de prédios em uma cidade virtual.



Fonte – Greuter et al. (10)

Figura 2 – Cidade gerada de forma completamente procedural.



Fonte – Greuter et al. (10)

Considerando isso, a utilização de agentes de software em GPC consiste justamente em controlar a aleatoriedade de forma granular. Mesmo que sejam baseados em regras de negócio simples, a interação entre o conjunto de agentes pode produzir resultados ricos em termos de conteúdo e diversidade. Nesse contexto, uma das técnicas mais estabelecidas para controlar a interação entre agentes são *steering behaviors*.

Utilizando vetores que descrevem forças e direções, tanto do próprio agente como de seus agentes vizinhos, *steering behaviors* foram criados para modelar comportamentos naturais complexos. Devido ao fato de serem baseados em cálculos de vetores locais, como dos agentes que estão próximos uns dos outros, as diferentes categorias de *steering behaviors* exploram essa temática de formas distintas.

O método *seek*, por exemplo, movimenta-se em direção a um ponto particular no espaço, seja ele fixo ou móvel. O método *flee*, em contrapartida, evita um ponto particular no espaço, criando vetores de força que direcionam o agente para longe desse local. A intensidade dos vetores de atração e repulsão, especificamente nesses casos, definem o quanto os agentes serão direcionados para perto ou para longe dos locais informados. Se tais agentes estiverem depositando sedimentos em uma simulação para geração de terreno, por exemplo, o resultado final será diferente com base na intensidade dos vetores força utilizados.

Steering behaviors possuem diversas características que podem criar um bom algoritmo de GPC. Eles são altamente parametrizáveis, seja no que se refere à intensidade das forças envolvidas, à quantidade de agentes ou aos seus tipos. Cada comportamento é baseado em regras simples, mas a interação entre diferentes agentes pode produzir resultados muito complexos e interessantes em termos de conteúdo. Além disso, operadores podem empiricamente adicionar ou remover agentes, alterar seus comportamentos e até mesmo ajustar as forças que os afetam, garantindo um certo controle da simulação sem abrir mão da aleatoriedade responsável pela diversidade do conteúdo gerado.

No contexto de GPC, uma técnica pode ser avaliada, dentre outros elementos, pela

quantidade de parâmetros disponíveis para customização, pela variação do conteúdo produzido e pelo controle sobre os resultados obtidos (17). Por consequência, a controlabilidade está diretamente ligada à variação do conteúdo, que, por sua vez, está relacionada à qualidade do resultado final.

Dessa forma, para um estúdio de desenvolvimento de jogos digitais, o controle sobre a aleatoriedade de uma técnica de GPC é um elemento primordial para garantir que essa ferramenta seja válida e útil no processo de criação. O controle do operador sobre o resultado final criado por uma técnica de GPC e a diversidade do conteúdo gerado são critérios chave na avaliação de uma técnica neste contexto (14).

Por fim, embora a utilização de agentes de software seja um tema recorrente em pesquisas, ainda não há muita literatura explorando o uso de *steering behaviors* nesse contexto. Aplicar uma abordagem como esta em um contexto de GPC, particularmente num conteúdo tão estudado e explorado como terrenos, é um tema relevante de pesquisa. Empregar GPC baseada em agentes de software e *steering behaviors* pode se tornar uma nova técnica de criação de conteúdo para estúdios de desenvolvimento de jogos num futuro não muito distante.

Se for mais pesquisada e difundida, esta nova técnica pode, além de auxiliar na produção de novos títulos, ajudar a reduzir custos e a refinar fluxos de criação já existentes. Estes fatores, por sua vez, poderiam aumentar as chances de sucesso e aceitação de muitos jogos publicados. Portanto, produzir e evoluir pesquisas nessa área pode ser frutífero a médio e longo prazo, o que confirma a relevância que este trabalho possui considerando os objetivos alcançados.

4 REVISÃO BIBLIOGRÁFICA

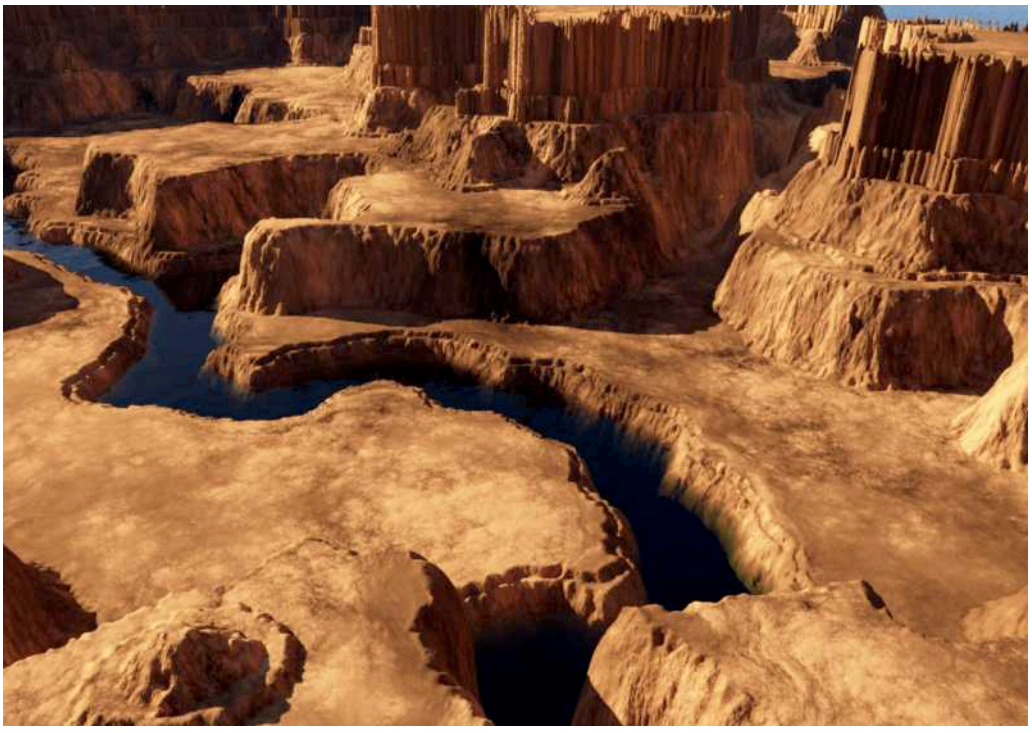
Este capítulo apresenta os conceitos e estudos que foram considerados chave para o desenvolvimento e concretização dos objetivos abordados por este trabalho. O foco será em elucidar as bases da GPC, especificamente para terrenos, e o uso e funcionamento de agentes de software e *steering behaviors*.

4.1 GERAÇÃO PROCEDURAL DE CONTEÚDO

O conceito de geração procedural de conteúdo (GPC) é dado pela utilização de qualquer técnica capaz de gerar uma ampla quantidade de conteúdo a partir de um processo descrito através de uma representação matemática. Em outras palavras, trata-se de um algoritmo que possui o papel de gerar conteúdo a partir de mecanismos de aleatoriedade aplicados a procedimentos matemáticos, dispensando a necessidade de uma equipe de profissionais para projetar e construir o conteúdo manualmente.

Considerando este fator, uma grande quantidade de conteúdo pode ser produzida em um curto espaço de tempo e com mínima intervenção. Por exemplo, um algoritmo de GPC voltado a criação de terrenos pode utilizar uma função matemática de ruído para gerar os pontos que compõem a paisagem. A Figura 3 apresenta o resultado de um terreno gerado através de um algoritmos de GPC. Note que o terreno gerado possui a configuração e o visual de cânions desérticos.

Figura 3 – Terreno produzido por GPC com foco na criação de cânions voltados a jogos digitais.



Fonte – de Carli et al. (6)

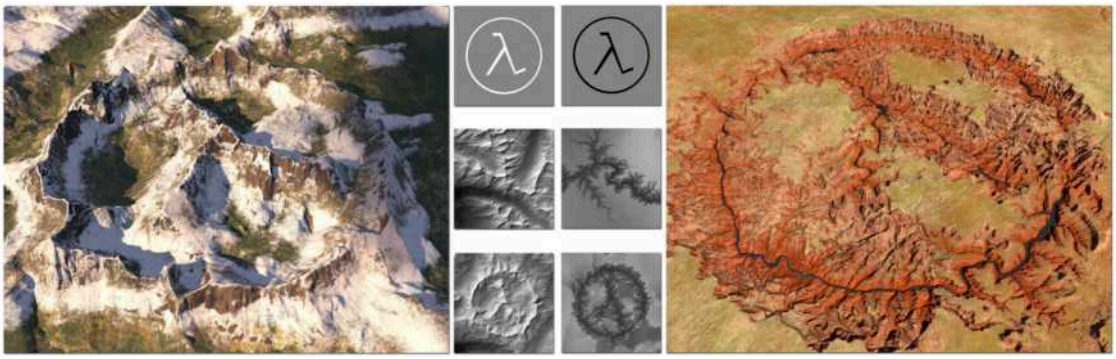
A GPC tem sido aplicada ao desenvolvimento de jogos digitais de diferentes formas, ainda que não possua ampla dedicação da comunidade acadêmica como campo de pesquisa (25). Desenvolver um jogo pode ser um trabalho árduo e, para Short; Adams (22), se não for feito um planejamento adequado, pode gerar gastos exorbitantes. Usar GPC para a criação de determinados conteúdos pode ajudar a economizar os recursos disponíveis no desenvolvimento de um jogo.

No entanto, mesmo que parte do conteúdo do jogo seja gerado através de uma técnica de GPC, ela ainda precisa ser devidamente contextualizada na temática do jogo em questão. Por exemplo, um jogo ambientado numa região desértica possui alguns requisitos a serem respeitados, como a escassez de águas e plantas. Realizar essa contextualização requer que as técnicas de GPC sejam parametrizáveis e controláveis a fim de manter todo o conteúdo do jogo alinhado com sua proposta, temática e mecânica.

Como mencionado, técnicas de GPC fazem uso de mecanismos de aleatoriedade para gerar o conteúdo. Enquanto este aspecto garante uma grande variabilidade do conteúdo gerado, também pode ser o causador de algumas inconsistências indesejadas. Segundo Short e Adams (22), GPC realmente é uma prática poderosa, mas que precisa ser usada com cautela, pois é capaz de salvar ou arruinar um projeto. Neste contexto, a parametrização de técnicas GPC se torna ainda mais necessária.

Com este propósito, De Carli et al. (7) define duas grandes categorias de técnicas de GPC aplicadas ao desenvolvimento de jogos: assistidas e não-assistidas. Uma técnica de GPC assistida exige que uma pessoa produza um material inicial que será utilizado como base pela técnica para geração de mais conteúdo. A Figura 4 apresenta um terreno gerado através de uma técnica de GPC assistida no qual o operador rabisca linhas que descrevem cadeias montanhosas. A partir dessa informação, a técnica produz a paisagem correspondente.

Figura 4 – Técnica de geração de terreno assistida

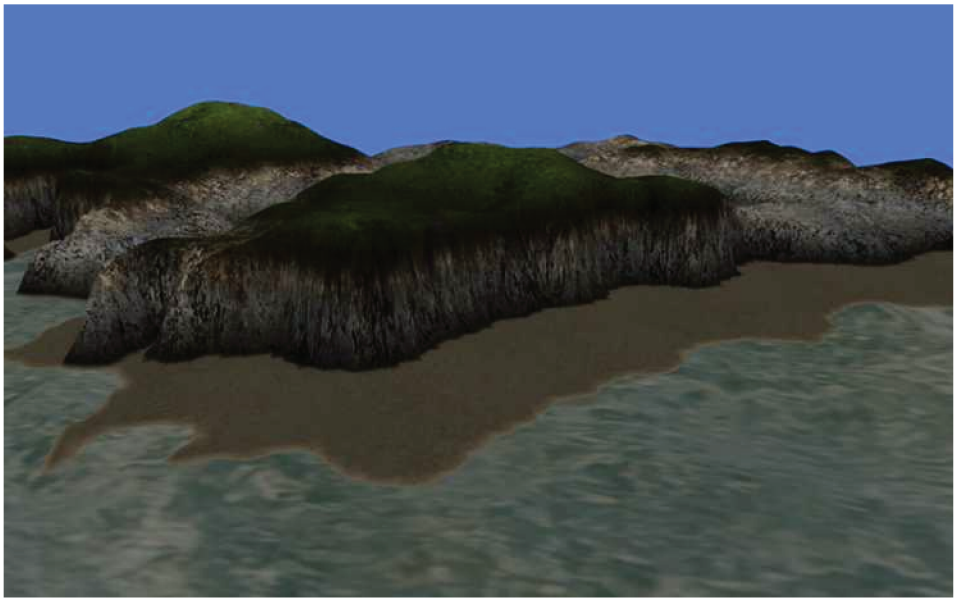


Fonte – Zhou et al. (29)

Em contraste, a técnica de GPC não-assistida é capaz de gerar conteúdo sem necessidade de intervenção ou interação com um operador. Nesse caso, a técnica utiliza um conjunto de regras e limitações internas, geralmente definidas no momento da criação da técnica, e itera sobre esses elementos para produzir conteúdo. A Figura 5 apresenta um terreno relativamente

complexo criado por GPC, o qual contém planícies, encostas e praias, tudo criado de forma completamente não-assistida.

Figura 5 – Técnica de geração de terreno não assistida



Fonte – Bevilacqua; Pozzer; d'Ornellas (2)

Ambas as técnicas assistidas e não-assistidas podem ser controladas por parâmetros gerais que limitam e controlam a geração de conteúdo, tais como o tempo de execução, variação de um elemento em específico, semente do algoritmo aleatório, dentre outros. No entanto, tudo depende da interação existente durante a execução da técnica. Ainda que ela possua suporte a alguns parâmetros, ela deve ser considerada como uma técnica não-assistida desde que estes parâmetros não sejam alterados durante a execução. Porém, se estes parâmetros forem alterados durante a execução, a técnica será considerada assistida.

A situação ideal para uma técnica de GPC é que ela seja não-assistida e tenha suporte a parâmetros que sejam capazes de facilmente controlar os resultados de forma eficiente e contextualizada. Esse aspecto reduz significativamente a necessidade de intervenção e interação, economizando recursos. Adicionalmente, Doran e Parberry (8) definem algumas propriedades que compõem um bom conteúdo gerado proceduralmente, as quais são apresentadas a seguir.

- Originalidade: conter elementos aleatórios e imprevisíveis, evitando duplicidade;
- Estruturação: conter estruturas bem definidas que não pareçam mera aleatoriedade;
- Atratividade: ser capaz de despertar o interesse com elementos aleatórios e estruturais;
- Agilidade: ser capaz de ser gerado de forma rápida;
- Controlabilidade: conter parâmetros intuitivos que possam controlar os resultados.

Além dessas propriedades, também há algumas classificações para conteúdos gerados de forma procedural. Togelius et al. (25) elaboraram e apresentaram algumas categorias que facilitam a distinção e compreensão de conteúdos gerados. Dentre as diversas classificações apresentadas, é possível destacar as apresentadas a seguir.

1. Online ou offline, que se refere ao tempo para produção de resultados;
2. Conteúdo necessário ou opcional, que se refere à possibilidade do operador complementar o resultado produzido pela técnica;
3. Baseado em uma semente de aleatoriedade ou em um vetor de parâmetros, que se refere à forma de decisão interna da técnica para limitar suas características aleatórias;
4. Geração de conteúdo estocástica ou determinística, que também se refere às características aleatórias da técnica, porém focadas na repetição de resultados;
5. Geração construtiva ou gera-e-testa, que se referem às características de geração de conteúdo ao longo do processo.

Essas propriedades e classificações são úteis ao analisar e avaliar conteúdos de forma mais quantitativa (25). Compreender os resultados é muito importante para realizar uma parametrização apropriada de conteúdos gerados proceduralmente. Dependendo do conteúdo, a controlabilidade pode ser um fator decisivo, especialmente se o conteúdo gerado for utilizado em jogos digitais.

Atualmente, existem muitos jogos que fazem uso ativo de técnicas GPC. Por exemplo, jogos com uma grande quantidade de fatores aleatórios, como *Rogue* e *Spelunky*, dependem da geração procedural para sua jogabilidade (22). Alguns jogos propõem um mundo infinito, como *No Man's Sky*, o que os torna dependentes da geração procedural para concretizar sua visão (22). Logo, Short; Adams (22) conclui que a utilização de técnicas GPC em um jogo depende diretamente do gênero escolhido para a mídia e sua proposta.

Além de economizar tempo de produção e desenvolvimento, o uso de GPC em jogos digitais pode ser um grande aliado em jogos geograficamente extensos (22). A geração procedural de terrenos é um tópico de grande importância, abordando desde ambientes naturais à ambientes urbanos. Segundo Smelik; Kraker; Groenewegen (23), este tópico tem sido muito explorado nas últimas quatro décadas e já fora aplicado para diferentes propósitos com êxito.

De acordo com Smelik et al. (24), essas técnicas são multidisciplinares e possuem uma certa aproximação dos processos de simulação computacional. Essa aproximação se daria pelo fato da geração se espelhar nos resultados obtidos em processos naturais, urbanos e físicos. Por exemplo, para gerar um terreno numa região tropical é necessário conhecimento sobre as características dessa região para que a o resultado seja coerente, como se o processo estivesse simulando uma região tropical.

Entretanto, embora este paralelo com a simulação possa ser feito, Galin et al. (9) afirmam que a geração de terrenos não tem relação direta com a simulação. Formas geográficas naturais possuem variedade e complexidade, sendo que a variação de planos, altitude e recursos podem estar presentes em uma mesma área, pois são consequências de longo tempo de eventos catastróficos, naturais ou intervenção humana. Portanto, é destacado que estratégias procedurais objetivam reproduzir os efeitos colaterais de tais fenômenos, e não simular o fenômenos. Por conta desses aspectos, ainda existem muitos desafios a serem revolvidos na área de modelagem de terrenos (9).

Considerando o que foi apresentado, Doran; Parberry (8) explicam que um dos desafios na geração procedural de um terreno é manter o equilíbrio entre parcelas consistentes e aleatórias. Um dos propósitos disso, segundo eles, é evitar duplicatas muito aparentes de conteúdo ao redor do terreno e manter o realismo no contexto geográfico. Para tal, faz-se necessária a existência de parâmetros que possam ajustar essas características a fim de gerar um terreno realista e coerente (8).

4.2 AGENTES

A definição de um agente já foi palco de muitas discussões e envolve uma série de divergências de opiniões, sendo assim, não há um definição amplamente reconhecida (26). Isso ocorre pela sua grande variedade de tipos, pela sua flexibilidade de utilização em diferentes aplicações e pelo fato de sua criação objetivar um propósito específico (13).

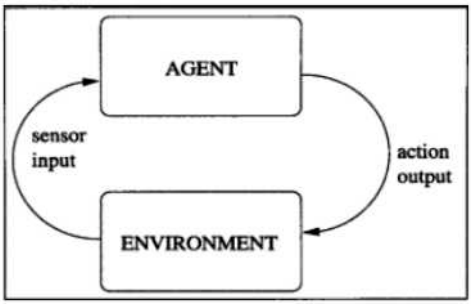
Correa Filho (5) afirma que um agente pode ser considerado uma entidade cuja operação se dá de forma contínua e autônoma, que trabalham com outros agentes e diversos processos no mesmo ambiente. Amandi (1) possui uma definição semelhante em que um agente é uma entidade computacional autônoma. Tal comportamento concede ao agente a habilidade de tomada de decisão sobre como agir perante as alterações no ambiente para atingir seus objetivos.

Ampliando essas definições, Shoham (21) estabelece que estados, intitulados estados mentais, são atribuídos aos agentes, tais como: decisões, objetivos, capacidades, compromissos e expectativas. Esses estados mentais permitem a ação dos agentes no ambiente, além de se assemelharem a características humanas.

No geral, todas as definições apresentadas tendem a apresentar a autonomia, a habilidade de ser operacional sem intervenção, como a ideia central e fundamental para a atividade de um agente (27). O trabalho de um agente é baseado nas informações que ele obtém do ambiente, chamada de entrada sensorial ou *sensor input*, e no retorno de ações que alteram o mesmo, chamado *action output* (27). Esse ciclo, composto da aquisição de informações e resultados de uma produção, é representado de forma simplificada na Figura 6.

Wooldridge (27) explica que um agente não dispõe do controle do ambiente por completo, mas tem a capacidade de influenciá-lo. Assim, em ambientes cujas circunstâncias parecem ser idênticas, a produção dos agentes pode obter diferentes resultados.

Figura 6 – Um agente e seu ambiente



Fonte – Wooldridge (27)

Diante disso, Wooldridge; Jennings (28) observaram que, além da autonomia, as variadas definições de agente dispõem de propriedades que os diferenciam dos demais objetos de um software, sendo elas:

- Habilidade social: possibilita que os agentes se comuniquem entre si, seja para solucionar um problema em comum ou auxiliar em uma tarefa;
- Reatividade: percepção e reação às mudanças no ambiente;
- Proatividade: a produção não depende da resposta do ambiente. Possuem a habilidade de realizar ações direcionadas a um objeto específico, podem decidir tomar a iniciativa (28).

A partir dessas propriedades, é possível afirmar que um sistema multiagente é dependente da coordenação entre seus agentes e suas produções. Jennings (12) expõe três motivos do porquê da necessidade de coordenação:

- As ações dos agentes são dependentes entre si;
- Necessidade de seguir restrições globais;
- Nenhum agente é autossuficiente, nenhum possui informações ou recursos para atingir os objetivos e resolver todo o problema sozinho.

Em outras palavras, todos os agentes são responsáveis pelo funcionamento e comunicação do sistema, em colaborar com seus objetivos visando a resolução do problema maior de forma conjunta (12).

Jennings (12) lista as vantagens que as características de sistemas multiagentes oferecem como um método de solução de problemas. Entre essas vantagens estão a velocidade de resolução por meio das produções em paralelo, a comunicação reduzida através do compartilhamento de dados parciais, flexibilidade e segurança, uma vez que quando um agente falha, outro assume suas responsabilidades.

Para a concepção de agentes e seu sistema, é indispensável a delimitação das habilidades e restrições dos agentes para garantir sua autonomia. Pozzer; Furtado; Ciarlini (18) categorizam agentes baseando-se na capacidade de produção e resolução de uma problemática. Essa categorização é composta por agentes reativos, cognitivos e híbridos.

Os agentes reativos reagem às modificações realizadas no ambiente e informações recebidas de outros agentes. Portanto, eles são influenciados apenas por regras já definidas e, como resultado, não são capazes de buscar possibilidades quando suas intenções se tornam incompatíveis com as circunstâncias em que o ambiente está inserido (18).

Ao contrário dos reativos, os agentes cognitivos são capazes de raciocinar, gerar e definir trabalhos para serem desenvolvidos conforme suas intenções (18). Porém, são incapazes de reagir rápido a imprevistos adequadamente.

Os agentes híbridos são definidos como uma solução das desvantagens apresentadas pelos agentes reativos e cognitivos. Eles realizam uma previsão de alto nível como os cognitivos, e os problemas menos críticos e significativas são solucionados como os reativos.

4.3 STEERING BEHAVIORS

Empregado principalmente em jogos digitais, *Steering behaviors* (SB) foram propostos por Reynolds (19) com o intuito de auxiliar a movimentação e comportamento de atores autônomos em um ambiente. Essa movimentação foi planejada para obter um resultado realista com a utilização de forças simples, que combinadas resultam em uma movimentação improvisada e interativa (19).

Se opondo ao que foi mencionado na seção anterior, apesar de ser uma técnica de movimentação de agentes, SB não utilizam de estratégias de planejamento ou previsão de eventos. Utilizam-se de informações locais, como obstáculos e forças dos atores próximos, para calcular suas interações e realizar a movimentação (3).

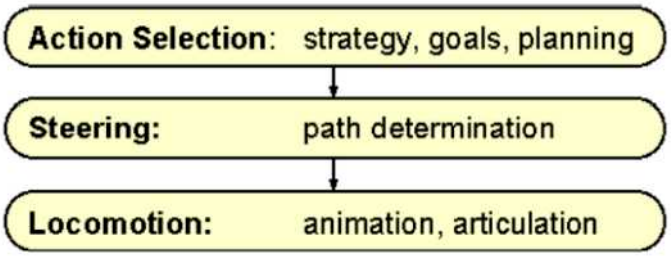
As forças podem ser descritas por vetores que alteram a direção e velocidade do ator, guiando-o para seu destino (4). Seguindo essa lógica, a posição de um agente é modificada a partir da velocidade, enquanto a velocidade é alterada pelas forças de direção nele aplicadas (19). Estes fatores englobam o comportamento e a possibilidade de ações que o ator pode realizar.

Sem a força de direção, o agente não teria a simulação de movimentações complexas e fluídas desejada. O ator seguiria uma linha reta para chegar ao seu destino, independente de obstáculos ou de atores ao redor (3).

Dessa forma, os agentes possuem um conjunto de regras determinadas de acordo com sua aplicação, objetivos, determinação e descrição do caminho a ser seguido. Com este contexto, Reynolds (19) propôs e descreveu sua visão utilizando de 3 camadas, ilustradas na Figura 7.

No topo da hierarquia proposta, encontra-se o *action selection* que representa os objetivos já definidos e o planejamento de cada agente. Logo abaixo, encontra-se a etapa de *steering* onde ocorre os cálculos das forças, de modo que os objetivos definidos na camada anterior sejam

Figura 7 – Hierarquia de 3 camadas em agentes de *steering behaviors*



Fonte – Reynolds (19)

atingidos. Por último, tem-se a etapa de *locomotion* que estabelece como o ator irá se mover do ponto atual para o destino.

A técnica de *steering behaviors* é classificada em tipos de comportamento específicos, tais como:

- *wander*: produz uma força que cria a impressão de que o agente está andando aleatoriamente, perambulando pelo ambiente (4);
- *seek*: direciona o agente a uma posição específica do ambiente (19);
- *flee*: ao invés de produzir uma força para conduzir o ator na direção do alvo, o *flee* cria a força resultante que afasta o ator (4);
- *arrive*: apesar de ser capaz em seguir uma direção, o *seek* não está qualificado para parar a sua movimentação suavemente. *Arrive* é o comportamento que permite que o agente diminua sua velocidade em direção ao alvo (4);
- *pursuit*: o alvo é móvel, portanto é necessário realizar uma previsão de onde o alvo estará e fazer ajustes conforme esse deslocamento ocorre (4);
- *evade*: semelhante ao *pursuit*, porém se afasta da estimada posição do alvo;
- *collision avoidance*: afasta o agente sempre que estiver muito próximo de colidir com um obstáculo.

5 TRABALHOS RELACIONADOS

O capítulo anterior apresentou os principais conceitos necessários para a compreensão e o desenvolvimento deste trabalho, incluindo GPC (geração procedural de terrenos), agentes inteligentes e *steering behaviors*. Todavia, cada um desses recursos pode ser utilizado de diferentes formas e com diferentes abordagens. Portanto, este capítulo objetiva apresentar alguns trabalhos que utilizam recursos GPC e agentes com propostas semelhantes ao deste trabalho, pois eles são essenciais para guiar e explicar metodologia adotada neste trabalho. Ambos os trabalhos estudam características reais para a formação de terrenos e utilizam de agentes inteligentes para usa modelagem.

5.1 CONTROLE SOBRE A GERAÇÃO PROCEDURAL DE TERRENOS UTILIZANDO AGENTES DE SOFTWARE

Com o objetivo de criar uma geração procedural de terrenos baseada em parâmetros naturais, Doran; Parberry (8) apresentam questões discutidas por outros autores e propõem uma abordagem para lidar com esse tema. A aplicação de agentes inteligentes foi justificada como solução para a questão de controlabilidade.

Doran; Parberry (8) consideraram os sistemas disponíveis no mercado, e concluíram que estes possuíam uma pequena quantidade de parâmetros de controle disponíveis para designers. Dessa forma, a utilização de agentes permitiria que o designer escolhesse a quantia e intensidade de diferentes características em um terreno, como montanhas e rios. Por consequência, o designer deteria o poder de escolha a partir da definição do tempo de vida de um agente e a quantidade de agentes de cada tipo. Sendo assim, a técnica desenvolvida é classificada como não assistida.

Para concretização do desejo de considerar fatores naturais e proporcionar maior controlabilidade ao designer, foram selecionados cinco elementos de modo que possibilitassem a criação de um terreno consistente e realista. Utilizou-se de agentes inteligentes para construção dos elementos no cenário, sendo eles: *coastline*, *smoothing*, *beach*, *mountains* e *rivers*.

De forma assíncrona, independente e simultânea, os agentes modificam o ambiente elevando pontos do nível do mar. Portanto, considerando que cada agente é criado com uma meta de pontos a serem gerados e uma direção desejada, o terreno é modelado de forma dinâmica pelos agentes.

No momento em que o processo é iniciado, todo o mapa está no nível do mar e os agentes *coastline* são os primeiros a agir. Eles são os responsáveis pela criação da linha que define área do terreno em que todos os demais agentes irão desempenhar suas tarefas. Esta etapa de execução é iniciada com apenas um agente, que em seguida "divide a tarefa a partir da criação de múltiplos agentes subordinados e os nomeando para trabalhar em determinadas partes do terreno"(8).

Figura 8 – Resultado dos agentes do tipo *coastline* com (da esquerda para direita) parâmetros de ação pequena, média e grande.

Fonte – Doran; Parberry (8)

Os próximos agentes na fila de execução são os encarregados da elaboração das áreas planas para formar regiões beira-mar. Os agentes *beach* se movimentam aleatoriamente nivelando o terreno com a linha costeira agindo como guia e garantindo que as praias não se tornem locais uniformes. Também é possível definir, como demonstrado na Figura 9, a largura das praia geradas entre pequena, média e grande. Esses agentes evitam áreas elevadas, ou seja, possuem o limite de altitude como um de seus parâmetros.

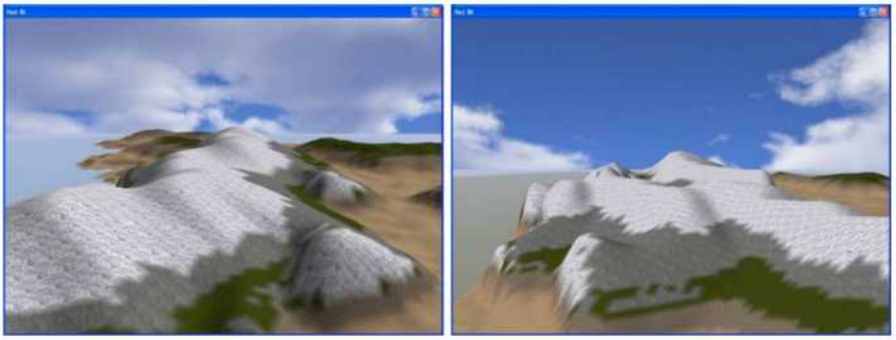
Figura 9 – Resultado dos agentes do tipo *beach* com as larguras pequena, média e grande.

Fonte – (8)

Os agentes que definem as elevações no terreno são os do tipo *mountain*. Doran; Parberry (8) elaboraram os agentes *mountain* com mais configurações do que os demais, pois estes tem o potencial de incorporar características mais interessantes ao terreno. Agentes *mountain* iniciam suas tarefas selecionando um ponto qualquer do mapa, um número máximo de altitude e uma direção inicial. Com a formação dos picos de montanhas em um V invertido, eles mudam sua posição dentro de 90º, formando um *ziguezague* e alteram sua direção ao chegar no oceano ou na borda do mapa. A inclinação do pico da montanha até sua base é estabelecida randomicamente de modo que compreenda os limites determinados pelo projetista, o que também contribui com a definição da largura das montanhas.

A suavização do terreno, realizada pelos agentes *smoothing* é aplicada após a elevação das montanhas. Enquanto a suavização é executada, é adicionado ruído em suas composições.

Figura 10 – Resultado dos agentes do tipo *mountain*



Fonte – (8)

Logo, os pontos de altitude são alterados para adicionar variância, singularidade e naturalidade sobre as arestas suavizadas e mudanças bruscas da altura no mapa.

Outro tipo de agentes que possuem uma importante função no sistema são os agentes *rivers*. Esses selecionam pontos aleatórios na linha costeira e nas montanhas para gerar cursos de água natural. Cada agente é capaz de criar um rio, isto é, o designer pode determinar a quantidade de rios desejada definindo do número de agentes. Porém, nas limitações definidas por Doran; Parberry (8), há a possibilidade de um agente não criar o rio caso enfrentem impedimentos.

Os agentes *rivers* foram desenvolvidos para evitar caminhos muito curtos e que não alcançam as montanhas. Com isso, são realizadas algumas tentativas para melhor posicionar o rio. Caso não encontrem uma possibilidade, desistem e consideram seu trabalho como concluído.

Doran; Parberry (8) constataram o sucesso de sua abordagem com resultados muito promissores que, segundo eles, prendem a atenção do jogador. Observou-se grande variedade nos mapas resultantes, mantendo o balanço entre a aleatoriedade latente e os parâmetros definidos pelo designer.

5.2 MODELAGEM PROCEDURAL DE ÁREAS URBANAS UTILIZANDO UM SOFTWARE BASEADO EM AGENTES

Assim como a pesquisa de Doran; Parberry (8), Lechner et al. (15) não pretendiam reproduzir locais já existentes. Buscavam desenvolver um modelo automatizado para criação de cidades convincentes, atraentes e realistas considerando sua época e cultura. O trabalho se concentrou em gerar e distribuir edificações urbanas no mapa, de modo que tanto aldeias quanto metrópoles aparentassem arquiteturas complexas e legítimas. Além disso, possibilitaram, a partir do uso de agentes inteligentes, que parâmetros pudessem ser inseridos para entregar maior liberdade artística, sendo obrigatória apenas a descrição do terreno.

Com a inserção da descrição do terreno, o sistema retorna um mapa com cinco tipos

principais de estruturas estudadas pelos autores, sendo eles: edifícios residenciais, comerciais, industriais, estradas e parques. A automatização da localização dessas estruturas urbanas é executada com a simulação da história do desenvolvimento e crescimento de um ambiente urbano. Diferentes épocas podem ser designadas para parcelas do mundo, por conseguinte, a simulação é enriquecida com variedade, modernismo e realismo.

Para cada tipo de edificação um agente foi encarregado, de maneira que todos os agentes interajam indiretamente entre si e diretamente com a sua parte do mundo. Assim, tornou-se possível diferentes interações realizadas pelo artista.

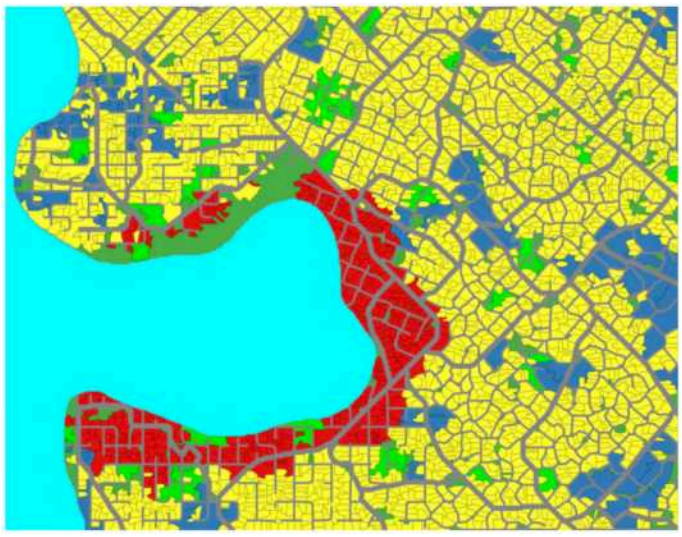
Classificando-se como uma técnica assistida, o artista pode escolher entre interagir com o mundo durante seu desenvolvimento automatizado, ou configurar parâmetros em segmentos específicos do mapa, ou ainda permitir que o processo seja concluído sem interrupções. O designer também é capaz de determinar o comportamento do sistema por uma interface de pintura.

A descrição do terreno base impacta na atividade de cada agente. Por exemplo, os agentes encarregados pelas áreas residenciais da cidade se posicionarão longe das áreas industriais e das estradas de alta velocidade. As áreas industriais serão maiores, menos populosas e seus agentes buscarão terrenos planos e de fácil acesso de transporte.

O desenvolvimento de estradas no sistema é o único que possui uma hierarquia, sendo os níveis mais altos compostos pelas estradas de alta velocidade e longa distância, e os níveis inferiores formados por ruas de acesso às áreas residenciais.

Na Figura 11, pode-se observar um dos resultados da geração procedural de uso de terras. As estradas estão destacadas em cinza, as áreas residenciais em amarelo, as áreas comerciais em vermelho, industriais em azul e parques em verde claro.

Figura 11 – Resultado da geração procedural do modelo de uma cidade



Fonte – (15)

A simulação da distribuição do uso de terras obteve o auxílio de uma colaboração

dos desenvolvedores do jogo SimCity da Eletronic Arts. A implementação foi realizada no NetLogo, uma linguagem de programação com simulação baseada em agentes. Lechner et al. (15) alcançaram padrões realistas na modelagem de cidades, tornando-as únicas, com a possibilidade de assistir seu desenvolvimento e moldá-lo ao seu favor.

6 IMPLEMENTAÇÃO

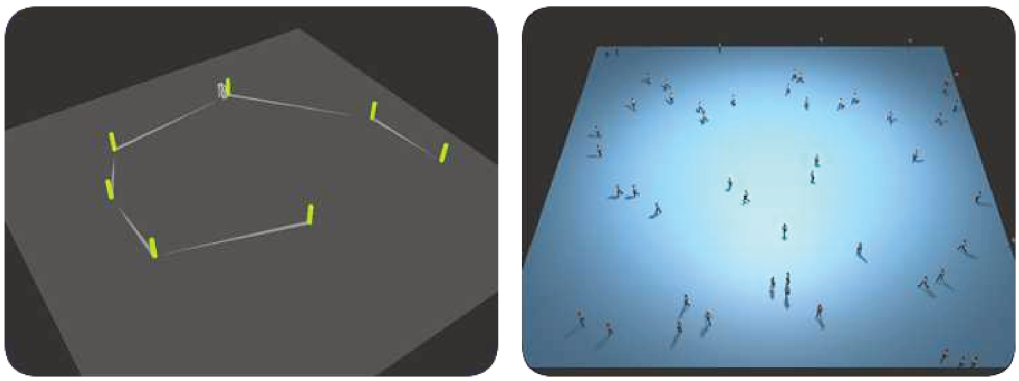
Esse capítulo apresenta a metodologia utilizada para a criação da técnica de GPC baseada em agentes de software guiados por *steering behaviors*. Adicionalmente, também são apresentadas as formas e critérios de avaliação dos resultados. Tanto a elaboração da técnica proposta quanto sua avaliação foram realizadas no contexto de geração de terrenos, visto que é um tema muito mencionado na literatura e cuja avaliação pode ser feita de forma mais holística.

O passo inicial desse trabalho está relacionado à criação de uma ferramenta que permita que diferentes mapas de altura sejam gerados a partir da interação de agentes de software guiados por *steering behaviors*. Visto que o foco era analisar os resultados produzidos pela técnica, e não na implementação ideal da técnica em si ou da visualização de seus resultados, ferramentas de código aberto foram utilizadas como ponto de partida.

Para a implementação deste projeto, foram utilizadas bibliotecas baseadas na *Three.js*[1], uma biblioteca que permite a criação e manipulação de cenários 3D com a praticidade *cross-browser*. Utilizando de um cenário 3D, torna-se mais fácil e prática a visualização das modificações realizadas pelos *steering behaviors* em um mapa de alturas.

No escopo relacionado aos agentes movidos por *steering behaviors*, utilizou-se da biblioteca *ThreeSteer*[2], cuja licença permite a utilização em projetos de pesquisa. Implementada por Eros Marcon, ela oferece classes e funções para criação de *steering behaviors* e também disponibiliza exemplos de sua aplicação. Por possuirem a *Three.js* como base, as classes do *ThreeSteer* permitem uma integração flexível com diferentes ferramentas externas. A biblioteca conta com uma aplicação gráfica das entidades de *steering behaviors* para a melhor compreensão de suas movimentações e características. A Figura 12 apresenta um exemplo dessa visualização.

Figura 12 – Exemplo da execução da biblioteca *ThreeSteer*



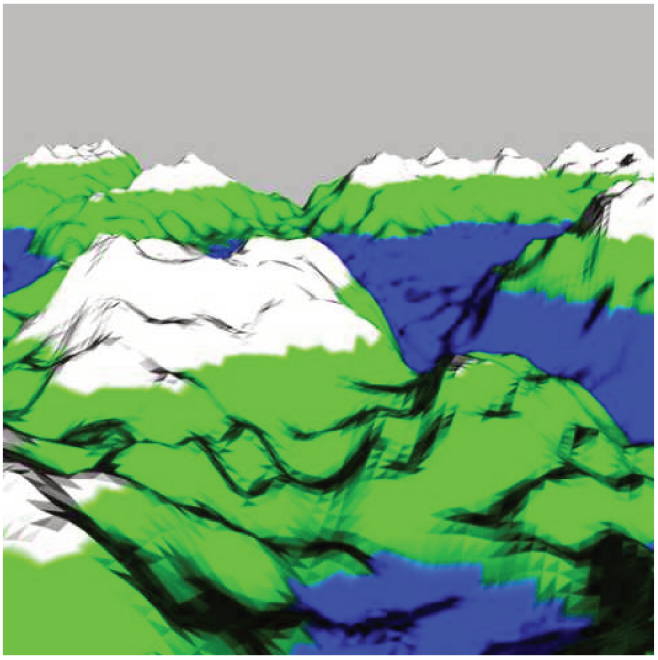
Fonte – Biblioteca *ThreeSteer*

Em relação à visualização dos resultados, especificamente do mapa de alturas produzidos pela técnica, foi proposta a visualização semelhante à utilizada pela biblioteca *ThreeSteer*. As-

[1] Disponível em `https://github.com/mrdoob/three.js/`
[2] Disponível em `https://github.com/erosmarcon/three-steer`

sim, usufruiu-se das funcionalidades de uma extensão de jogos da Three.js, a `ThreeX.Terrain`[3], cuja licença também permite a utilização em projetos de pesquisa. Conforme mostrado em sua documentação, a extensão gera um terreno de forma procedural com Ruído Simplex e Ruído de Perlin, podendo mostrar o terreno em 2D ou 3D conforme o desenvolvedor desejar. A Figura 13 ilustra a visualização de um terreno através dessa ferramenta.

Figura 13 – Exemplo da visualização da extensão *ThreeX.Terrain*



Fonte – Extensão *ThreeX.Terrain*

Embora apenas o último elemento apresentado esteja relacionado com uma técnica de geração de terrenos, pretendeu-se criar uma ferramenta que pudesse ser utilizada como base no futuro por outros pesquisadores para a continuação dos trabalhos. A estratégia científica de avaliação escolhida para o presente trabalho foi a pesquisa experimental (16). Essa abordagem é composta por um conjunto de testes controlados e manipulação de variáveis utilizados para se entender processos de causa e efeito (20). Nesse contexto, um experimento é descrito como a manipulação de uma variável ou um conjunto delas, e a medição de quaisquer mudanças que isso possa ocasionar em outras variáveis do experimento.

Com base nessa análise de causa e efeito, pode-se estabelecer qual a relação e comportamento de uma variável dependente, e a técnica de GPC baseada em agentes de software e *steering behaviors*, no caso desse trabalho. Em outras palavras, cada agente inserido na simulação terá um comportamento de movimentação estipulado por *steering behaviors*, como *seek*, e um comportamento de produção de conteúdo. Para estes experimentos, o comportamento de produção de conteúdo se refere a depositar e remover sedimentos em um mapa de altura, isto é, incrementar ou decrementar o valor de um ponto (i, j) em um *heightmap* relacionado com a posição (x, y) atual do agente.

[3] Disponível em `https://github.com/jeromeetienne/threex.terrain`

Os experimentos foram compostos por diferentes rodadas de experimentação, cada uma com diversos tipos de *steering behaviors*. Nesse contexto, é importante destacar que para alcançar os objetivos estabelecidos, foi necessário realizar alterações em ambas as bibliotecas. No código principal do projeto, foram inseridas algumas constantes essenciais, sendo elas: largura e profundidade máximas do plano, número de entidades de um tipo X de *steering behaviors*, quantidade de iterações, e iteradores do mapa de altura.

O mapa de alturas foi definido como um objeto que, além de armazenar a matriz que define a altura de cada ponto no mapa, também possui a altura mínima e máxima que um ponto pode atingir, largura e profundidade, sendo essas iguais as estabelecidas para o plano 3D. O mapa de alturas oferece as seguintes variáveis e funções:

- `allowUpdate` - variável que determina se o mapa de alturas pode ou não ser atualizado. Padrão *true*;
- `update` - função que atualiza os valores do mapa de alturas. Para isso, é verificado se é possível atualizar o mapa a partir do `allowUpdate`, então é calculada a posição do *steering behavior* no mapa de alturas e a última posição alterada pelo mesmo;
- `constructMap` - função que inicializa a matriz do mapa de alturas;
- `showMap` - quando chamada, essa função mostra o mapa de alturas no console;
- `denyUpdate` - função que altera a variável allowUpdate para *false*.

Para a adaptação da *ThreeSteer*, foram necessárias pequenas alterações, como a definição de um id único para cada *steering behavior* criado e a criação de um vetor que salvasse a última posição do mapa de alturas alterada pela entidade.

Tendo em vista que o projeto descrito consiste na criação de um mapa de alturas, seria contra-produtiva a utilização do Ruído de Perlin implementado na extensão *ThreeX.Terrain*. Dessa forma, o algoritmo foi modificado para aceitar o mapa de alturas originado das ações dos *steering behaviors*. Ou seja, a extensão passou a ser responsável por mostrar o mapa de alturas na tela conforme os parâmetros e cores contidos na extensão.

Considerando que o plano utilizado na biblioteca *Three.js* é escalável, o máximo de altura foi definido pelo valor de 0.7. A escolha do valor foi influenciado pelas limitações da extensão *ThreeX.Terrain* e pela funcionalidade de mudança de coloração de acordo com valor da altura dos pontos, considerando principalmente a melhor visualização dos resultados adquiridos.

Com os ajustes realizados, foi possível iniciar o planejamento dos experimentos a serem executados. Como primeiro passo, deveriam ser definidos os *steering behaviors* a serem utilizados e combinações entre eles. Analisando cada tipo, pôde-se concluir quais seriam mais promissores a gerar um mapa de alturas consistente as expectativas. Assim, dividiu-se os experimentos em fases:

- Fase 1 - consiste na execução de diversos agentes de apenas um tipo de *steering behaviors* por vez no plano. Os agentes poderão apenas elevar os pontos do mapa de alturas;
- Fase 2 - consiste na execução de dois tipos de *steering behaviors* por vez que interajam entre si ou não. Nessa fase, os agentes também poderão apenas elevar os pontos do mapa de alturas;
- Fase 3 - consiste na execução de dois tipos de *steering behaviors* por vez, podendo interagir entre si ou não. Nessa fase, metade de agentes poderão apenas elevar os pontos, enquanto os demais poderão apenas abaixar os pontos do mapa de alturas.

Para a realização dos experimentos, definiu-se um plano 500 por 500 *pixels*, com as propriedades padrão de cada *steering behaviors* definida pela biblioteca *ThreeSteer*. Os experimentos foram realizados com o período máximo de 20 mil iterações da função principal *animate* da *Three.js* por teste. Os coeficientes testados foram a quantidade de agentes no plano e o fator de elevação dos ponto do mapa de alturas. Foram testados:

- 10 agentes de cada tipo ou combinação com elevação de 0.05, 0.1 e 0.2;
- 20 agentes de cada tipo ou combinação com elevação de 0.05, 0.1 e 0.2;
- 40 agentes de cada tipo ou combinação com elevação de 0.05, 0.1 e 0.2.

Esses fatores foram considerados para todas as três fases dos experimentos. Com isso, torna-se perceptível que todos os parâmetros criados para a realização deste trabalho são definidos antes da execução do algoritmo. Por conseguinte, não há a interação de um operador com o programa durante a geração do mapa de alturas, ou seja, a técnica apresentada se classifica como não assistida.

É importante destacar que, embora a validação experimental esteja sendo utilizada nesse estudo, não há uma hipótese sendo validada em cada uma das fases descritas. Isso porque a utilização de agentes de software, guiados por *steering behaviors*, não possui um resultado determinístico assim como a geração por ruído de Perlin. Logo, a validação experimental visou explorar a viabilidade e os desdobramentos visuais, por exemplo conteúdo, da utilização desses agentes.

6.1 FASE 1

Como definido, para realizar os experimentos da primeira fase, consideraram-se os tipos de *steering behaviors* que possuíam menor dependência de fatores externos e interações para realizar sua movimentação.

Como visto nos capítulos anteriores, o *steering behavior Seek* precisa um elemento para perseguir, sendo ele fixo ou móvel. Ou seja, podem-se definir pontos aleatórios ou fixos no

plano para que o *Seek* pudesse agir. Porém, um dos objetivos é utilizar unicamente *steering behaviors* para a construção do mapa de alturas e, como o *seek* requer a existência de pontos fixos ou móveis adicionais para que ele funcione, concluiu-se que não entraria para a primeira fase de testes.

O tipo *Pursue*, por possuir características similares às apresentadas pelo tipo *Seek*, também não poderia entrar na primeira fase de testes, tendo a necessidade de outro agente para ser executado. Outro tipo que também foi considerado para essa fase de experimentos foi o *Arrive*. No entanto, teve-se novamente o empecilho que os dois tipos anteriores apresentaram: a demanda pela existência de um ponto destino. No caso do *Arrive* e do *Seek*, se isso fosse feito, ambos os agentes elevariam os valores do mapa de alturas sempre em linha reta em direção ao ponto destino. A alteração do mapa de alturas em linha reta não é interessante para esse trabalho, uma vez que objetivo é a criação de um ruído que possa tornar-se um terreno a partir do processo de smoothing.

Pode-se citar ainda a análise realizada sob os tipos *Evade* e *Collision Avoidance*. Ambos poderiam ser utilizados como uma maneira de evitar as limites do plano. Como consequência, os pontos altos do mapa de alturas ficariam mais concentrados no centro, impedindo assim que prováveis cadeias de montanhas se formassem em suas bordas. Mais uma vez, esses agentes necessitam de interações para se movimentarem, para evitar bordas e até outros agentes. Dessa forma, não seria viável os inserir na primeira fase de testes.

Por fim, o único tipo do *steering behaviors* que não exigia a existência de outros para realizar sua movimentação era o tipo *Wander*. Este se demonstrou ser o mais promissor para realização da primeira fase de testes justamente por não demandar recursos adicionais e ser mais independente.

6.2 FASE 2

Considerando que a fase anterior utilizou agentes que não exigiam a coexistência com outros tipos de agentes, a segunda fase compõe de uma combinação de dois tipos de agentes de *steering behaviors* diferentes. Utilizou-se o tipo *Wander* da fase 1 com outros tipos de *steering behaviors*. Por exemplo, cada agente com comportamento de tipo *Seek* teria um agente com comportamento do tipo *Wander* para perseguir.

Para esta fase foram executadas as seguintes combinações de *steering behaviors*:

- *Wander* e *Seek* - onde os agentes *Seek* perseguem os agentes *Wander*. Os agentes *Seek* possuem metade da velocidade dos agentes *Wander*;
- *Wander* e *Pursue* - onde os agentes *Pursue* perseguem os agentes *Wander*. Os agentes *Pursue* possuem metade da velocidade dos agentes *Wander*;
- *Wander* e *Collision Avoidance* - onde *Collision Avoidance* é aplicado em todos os agentes *wander*;

- *Wander* e *Flee* - onde *Flee* é aplicado a todos os agentes *wander*, fazendo com que eles se afastem em ordem circular. Ou seja, a última entidade criada se afasta da primeira, e da primeira da segunda, assim por diante.

6.3 FASE 3

Uma vez que as fases 1 e 2 estejam concluídas, a terceira fase se apresenta como uma exploração dos resultados das fases anteriores. Por conta disso, os experimentos da terceira e última fase apresentam os tipos e combinações de *steering behaviors* que mostraram resultados com maior variação de alturas em comparação com as fases anteriores.

Além disso, nessa fase metade dos agentes criados terão como responsabilidade aumentar os valores dos pontos do mapa de alturas, enquanto a outra metade diminui os valores. O valor definido para a remoção de sedimentos do mapa de alturas foi a metade do valor de adição. Dessa forma, um agente não desfaz por completo a alteração realizada por outro ao passar sobre o mesmo ponto.

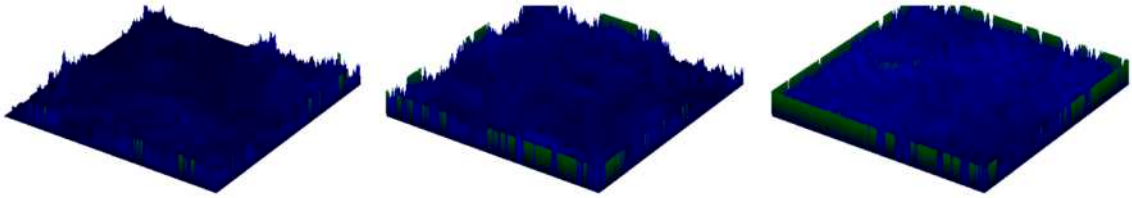
7 RESULTADOS

Este capítulo apresenta os resultados obtidos a partir dos experimentos descritos no capítulo anterior. Com os resultados, permitiu-se a análise e discussão sobre a utilização de agentes de *steering behaviors* para a criação de um mapa de alturas, e por consequência, a geração procedural de um terreno.

7.1 FASE 1

Na fase 1, como exposto no capítulo anterior, era necessária a aplicação de um tipo de *steering behavior* que não exigisse a presença de outros para executar seu padrão de movimentação. Dessa forma, os experimentos foram executados com a utilização do *steering behavior Wander*. É importante recordar que todos os experimentos foram executados com 10, 20 e 40 agentes de cada tipo ou combinação, e com iteradores do mapa de alturas com os valores de 0.05, 0.1 e 0.2. Em linhas gerais, as variáveis alteradas foram quantos agentes existiam no mapa em um dado momento e o quanto de sedimentos esses adicionavam ou subtraiam das posições por onde passavam.

Figura 14 – Resultados gerados pelos experimentos com iterador 0.05. Da esquerda para a direita, com 10, com 20 e com 40 agentes do tipo *Wander*.

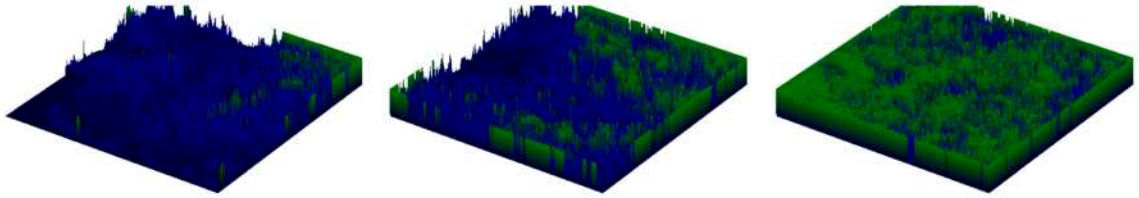


Fonte – Autora

Considerando que o plano utilizado na biblioteca *Three.js* é escalável e que seu máximo de altura tenha sido definido pelo valor de 0.7, os experimentos com o iterador do mapa de alturas como 0.05 mostraram uma boa variação de alturas, mesmo que nunca atingisse valores muito altos. A Figura 14 ilustra o resultado.

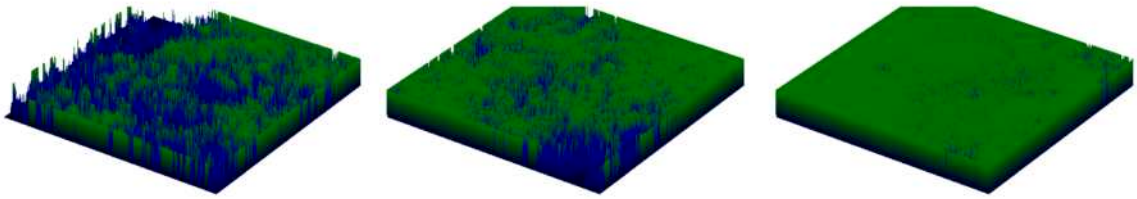
É possível observar nos experimentos com 20 e 40 agentes, porém, que as bordas do plano se encontraram mais altas que as demais áreas. Isso ocorre devido à movimentação dos agentes *Wander* que, ao se aproximarem demais das bordas, podem acabar colidindo com eles mesmos diversas vezes, aumentando a altura dos mesmos pontos múltiplas vezes. Esse acontecimento pode ser uma característica indesejada de se obter em um mapa de alturas dependendo do objetivo do projeto. Entende-se que não permitir que os agentes *Wander* se aproximem dos limites do mapa de altura, é impedir que futuras cadeias montanhosas sejam formadas naquelas regiões. Diversos jogos digitais utilizam de terrenos muito altos como uma estratégia para impedir que os jogadores encontrem o fim do mapa desenvolvido.

Figura 15 – Resultados gerados pelos experimentos com iterador 0.1. Da esquerda para a direita, com 10, com 20 e com 40 agentes do tipo *Wander*.



Fonte – Autora

Figura 16 – Resultados gerados pelos experimentos com iterador 0.2. Da esquerda para a direita, com 10, com 20 e com 40 agentes do tipo *Wander*.



Fonte – Autora

Os experimentos com iterador em 0.1 mostraram, como esperado, maior variação de alturas contendo regiões mais altas que as apresentadas na imagem 14. Apresentaram-se alturas incoerentes nas bordas do mapa com maior frequência, além de, no experimento com 40 agentes, majoritariamente não permitir a criação de áreas perfeitamente planas, pois, mesmo com alturas próximas, elas apresentavam variações.

Por possuírem um iterador mais próximo do limite de altura do mapa, os experimentos com iterador em 0.2 demonstraram menor variação. Observa-se que quanto maior o número de agentes *Wander* moldando o mapa de alturas, mais rápido alcançava o limite de altura do mapa, como apresentado no experimento com 40 agentes, ilustrado na Figura 16. Com isso, pode-se especular que a indicação de que quanto mais perto o iterador estiver do limite de alturas, menos promissor será o resultado do mapa de alturas para a construção de um terreno.

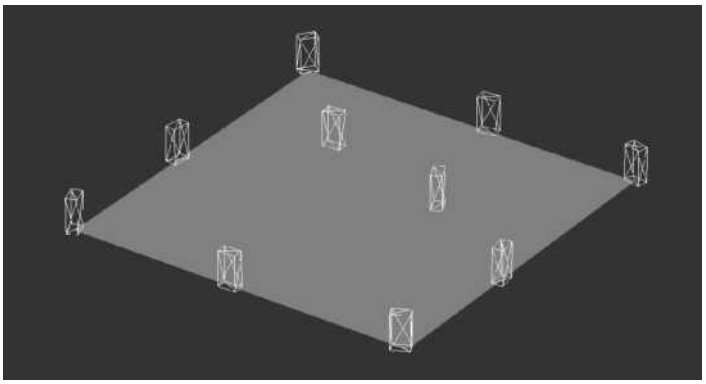
A conclusão da primeira fase de experimentos torna visível a diferença de resultado do mapa de alturas quando exposto a uma quantidade maior ou menor de agentes de determinado tipo de *steering behaviors*. Também foi possível perceber que o iterador impacta efetivamente a geração do mapa de alturas de um terreno, podendo ser ajustados para cada caso.

7.2 FASE 2

Na segunda fase de execução dos experimentos, significativas observações puderam ser realizadas. É importante salientar que os testes com as combinações de *steering behaviors*

Wander e *Collision Avoidance*, e *Wander* e *Flee*, foram descartados para a fase 3 por questões executivas.

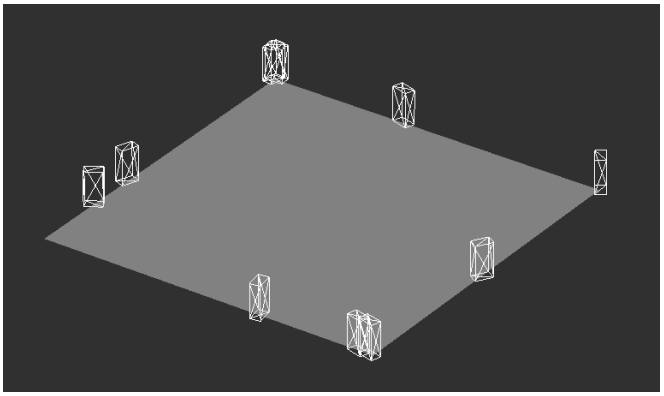
Figura 17 – Experimento da combinação *Wander* e *Collision Avoidance*.



Fonte – Autora

Para a implementação da combinação *Wander* e *Collision Avoidance*, foi aplicado e determinado que todos os elementos de tipo *Wander* deveriam evitar de colidir entre si. Em um plano com poucos *steering behaviors*, essa técnica funciona tão bem quanto os experimentos citados na seção anterior com dois ou três agentes, porém, quando são inseridos muitos agentes para modificar o mapa de alturas, eles se movem tão rapidamente que acabam por apresentar um comportamento muito similar a um ponto fixo no mapa. Isto é, como todos os agentes evitam colidir entre si, eles se dispõem em uma determinada distância dos demais. Considerando ainda que um agente está cercado por muitos outros, o agente *Wander* permanece fixo em um ponto girando, ou "perambulando", ao redor do seu próprio eixo. Este comportamento pode ser verificado na Figura 17.

Figura 18 – Experimento da combinação *Wander* e *Flee*.



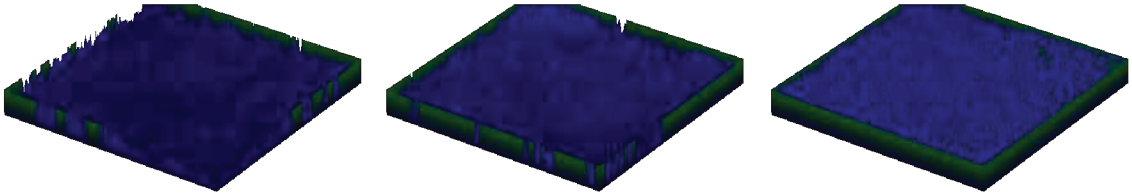
Fonte – Autora

A combinação *Wander* e *Flee* foi aplicada de modo circular e, portanto, para todo agente de tipo *Wander* o método *Flee* era empregado. Por exemplo, em um experimento de 10 agentes que vagam pelo mapa, o agente 1 iria se afastar do agente 2, o agente 2 se afastaria do agente

3 assim por diante. Por mais que esta abordagem pudesse parecer prover um mapa de alturas favorável para a criação de um terreno, após alguns segundos do início do experimento, os agentes se afastavam uns dos outros e do centro. Por conseguinte, os agentes circulavam as bordas do plano sem perspectiva de retornarem ao centro, ou seja, o resultado da combinação *Wander* e *Flee* apenas frisa o aumento excessivo do valor dos pontos nos limites do mapa citado na seção anterior.

Por sua vez, a aplicação da combinação dos tipos *Wander* e *Seek* foi realizada com os tipos de *steering behaviors* definidos para agentes diferentes. Sendo assim, cada agente *Wander* possui um outro agente *Seek* para segui-lo em seu trajeto. Os agentes *Seek* completam seu percurso com a metade da velocidade dos agentes *Wander*. Os resultados obtidos com essa implementação são ilustrados na Figura 19.

Figura 19 – Resultados gerados pelos experimentos com iterador 0.05. Da esquerda para a direita,com 10, com 20 e com 40 agentes do tipo *Wander* e *Seek*.

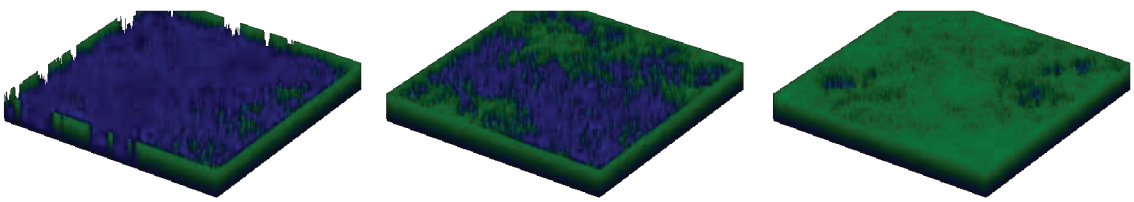


Fonte – Autora

Observou-se que o padrão do mapa de alturas adquirido pela combinação desses *steering behaviors* se aproximou dos resultados encontrados na fase 1 dos experimentos. Ele apresentou, porém, menos variação entre as alturas finais. Quanto maior o iterador de sedimentos e quanto maior o número de entidades disponível para os manipular, menor é a variação de alturas.

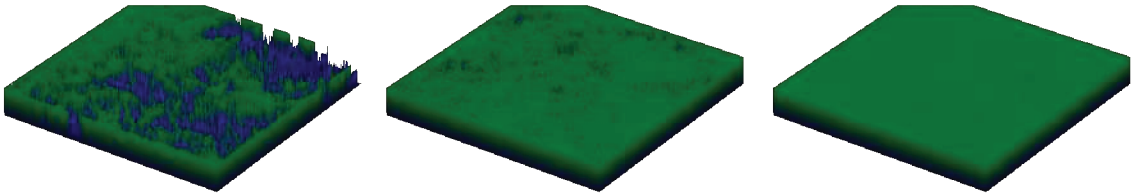
Este padrão de variação pode ser melhor analisado a partir dos experimentos com iterador de sedimentos em 0.1 e 0.2, representados pelas Figuras 20 e 21. Os resultados apresentam padrões de altitudes quase sem variação e muito próximas ou atingindo a altura máxima permitida.

Figura 20 – Resultados gerados pelos experimentos com iterador 0.1. Da esquerda para a direita, com 10, com 20 e com 40 agentes do tipo *Wander* e *Seek*.



Fonte – Autora

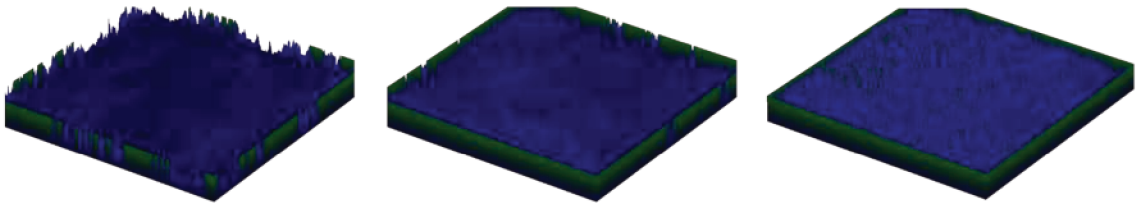
Figura 21 – Resultados gerados pelos experimentos com iterador 0.2. Da esquerda para a direita,com 10, com 20 e com 40 agentes do tipo *Wander* e *Seek*.



Fonte – Autora

Por possuir mais agentes trabalhando de uma só vez, essa combinação apresentou maior inserção de sedimentos nas laterais do plano. Este fato pode estar relacionado a quantidade de entidades presentes no experimento e ao seu tipo, uma vez que o *steering behavior Seek* demonstra um padrão de movimentação semelhante ao *steering behavior* do qual é dependente.

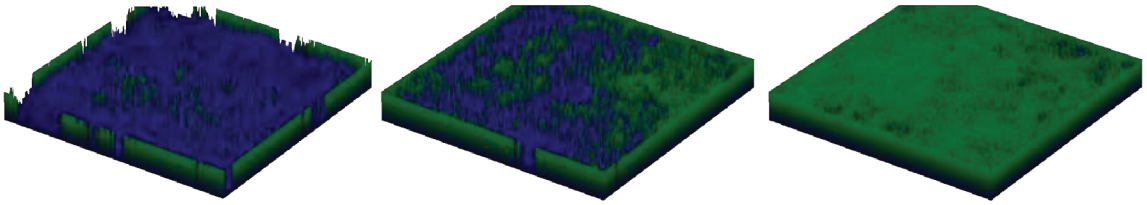
Figura 22 – Resultados gerados pelos experimentos com iterador 0.05. Da esquerda para a direita,com 10, com 20 e com 40 agentes do tipo *Wander* e *Pursue*.



Fonte – Autora

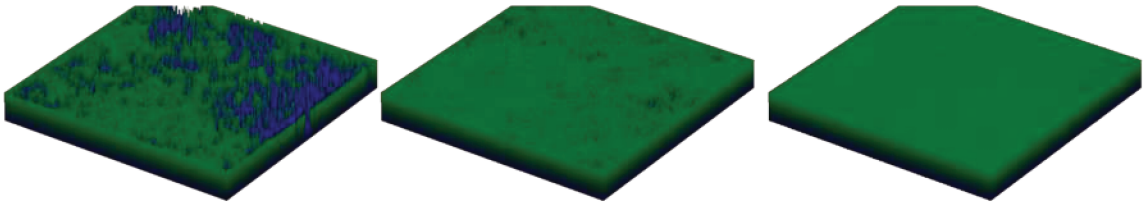
Com a combinação entre os agentes *Wander* e *Pursue*, foram produzidos resultados com variação maior do que *Wander* e *Seek*, e mais próximos dos resultados da primeira fase dos experimentos, como pode ser constatado nas Figuras 22, 23 e 24. Porém, apesar do resultado ser propício à criação de um terreno, percebeu-se durante a execução um padrão não favorável à criação de conteúdo na movimentação da entidades com o tipo *Pursue*.

Figura 23 – Resultados gerados pelos experimentos com iterador 0.1. Da esquerda para a direita,com 10, com 20 e com 40 agentes do tipo *Wander* e *Pursue*.



Fonte – Autora

Figura 24 – Resultados gerados pelos experimentos com iterador 0.2. Da esquerda para a direita,com 10, com 20 e com 40 agentes do tipo *Wander* e *Pursue*.



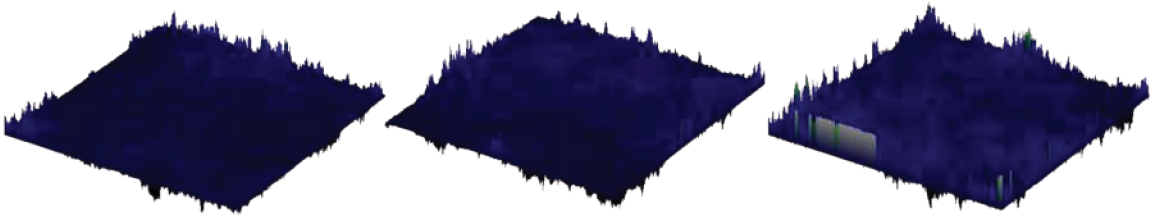
Fonte – Autora

Por realizar a mesma movimentação de perseguição de elementos, como o *Seek*, com um fator de previsão de onde o elemento estará, a movimentação das entidades *Pursue* se transformam no mesmo padrão de movimentação do *steering behavior* do tipo *Wander*. Isto é, em um plano que teríamos 10 agentes *Wander* e 10 agentes *Pursue* depositando sedimentos no mapa de alturas, teríamos o equivalente a 20 agentes *Wander*. Com base nessa constatação, a combinação dos *steering behaviors Wander* e *Pursue* não avançou para a terceira fase do experimento.

7.3 FASE 3

A terceira fase dos experimentos desse trabalho foi construída com as combinações remanescentes das fases anteriores. Sendo assim, nessa última fase os experimentos executados consistiram de duas aplicações: entidades com o *steering behavior* do tipo *Wander*; e entidades com o *steering behavior* do tipo *Wander* acompanhados do mesmo número de entidades do tipo *Seek*.

Figura 25 – Resultados gerados pelos experimentos com iteradores 0.05 e -0.025. Da esquerda para a direita, com 10, com 20 e com 40 agentes do tipo *Wander*.

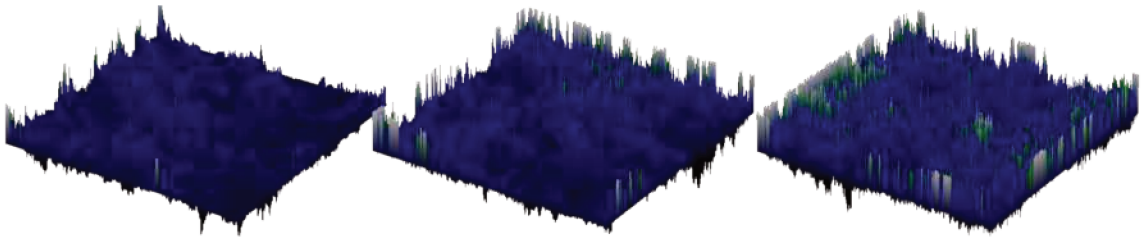


Fonte – Autora

Ao mesmo tempo, o diferencial dessa fase de experimentos está nos iteradores de sedimentos no mapa de alturas. Enquanto nas fases 1 e 2 sedimentos eram inseridos por todos os agentes na mesma proporção, na fase 3 metade dos agentes adicionaram sedimentos ao mapa de alturas, ao passo que a outra metade dos agentes removia sedimentos. Como citado no capítulo anterior, a remoção dos sedimentos foi realizada com um valor equivalente à metade do iterador.

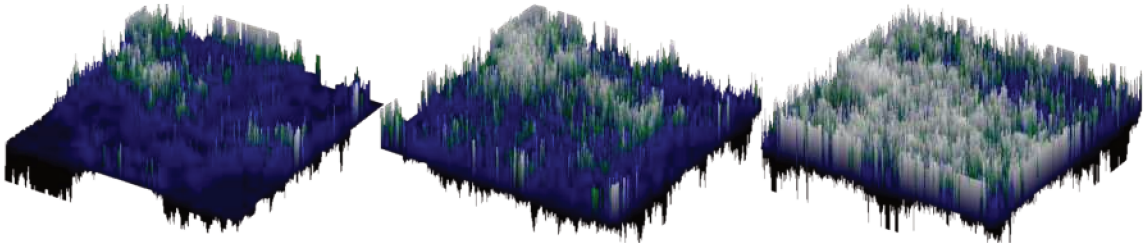
Com isso, a remoção de sedimentos não excluiria completamente os efeitos da adição no mapa de alturas final.

Figura 26 – Resultados gerados pelos experimentos com modificadores 0.1 e -0.05. Da esquerda para a direita, com 10, com 20 e com 40 agentes do tipo *Wander*.



Fonte – Autora

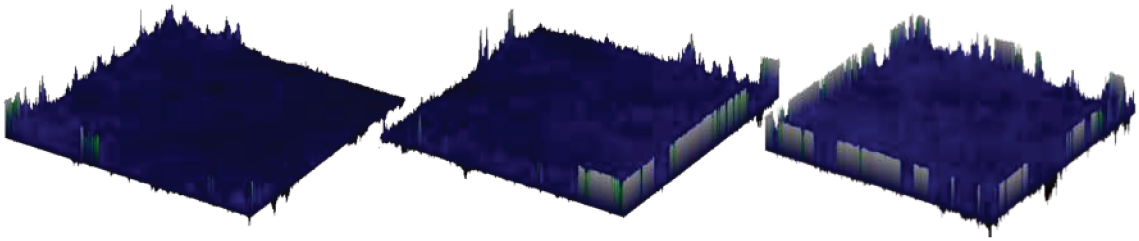
Figura 27 – Resultados gerados pelos experimentos com modificadores 0.2 e -0.1. Da esquerda para a direita, com 10, com 20 e com 40 agentes do tipo *Wander*.



Fonte – Autora

Os experimentos com entidades com tipo de *steering behavior Wander*, com modificadores de 0.05 e -0.025, mostram resultados muito semelhantes aos vistos durante a primeira fase de testes. Na Figura 25, é possível verificar que, devido ao baixo valor dos iteradores, formaram-se alturas menores que podem ser consideradas para a formação de terrenos de baixa altitude.

Figura 28 – Resultados gerados pelos experimentos com modificadores 0.05 e -0.025. Da esquerda para a direita, com 10, com 20 e com 40 agentes do tipo *Wander* e *Seek*.

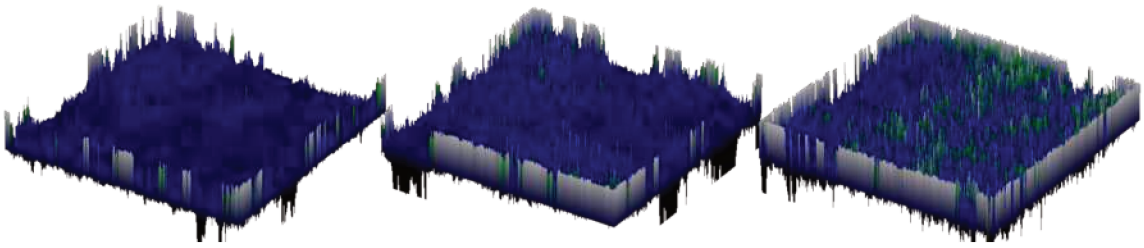


Fonte – Autora

A variação de alturas se manteve muito presente nos experimentos com *Wander* e são facilmente analisadas com os modificadores 0.05 e -0.025, e 0.1 e -0.05. Com modificadores

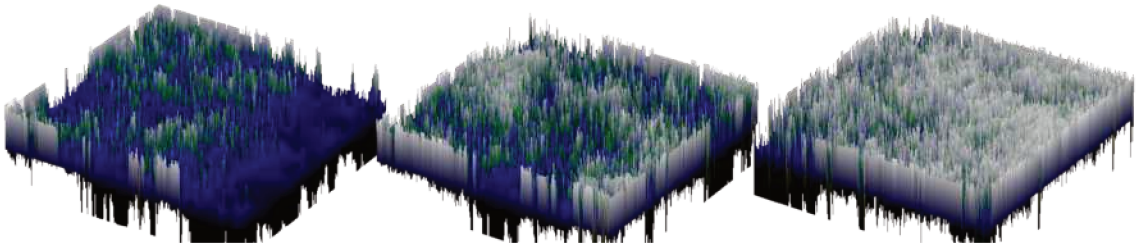
0.2 e -0.1, a variação se tornou mais evidente em comparação a primeira fase. Os resultados produzidos apresentaram um potencial de formação de terrenos essencialmente altos, como uma cadeia montanhosa. Esse efeito pode ser verificado na Figura 27.

Figura 29 – Resultados gerados pelos experimentos com modificadores 0.1 e -0.05. Da esquerda para a direita, com 10, com 20 e com 40 agentes do tipo *Wander* e *Seek*.



Fonte – Autora

Figura 30 – Resultados gerados pelos experimentos com modificadores 0.2 e -0.1. Da esquerda para a direita, com 10, com 20 e com 40 agentes do tipo *Wander* e *Seek*.



Fonte – Autora

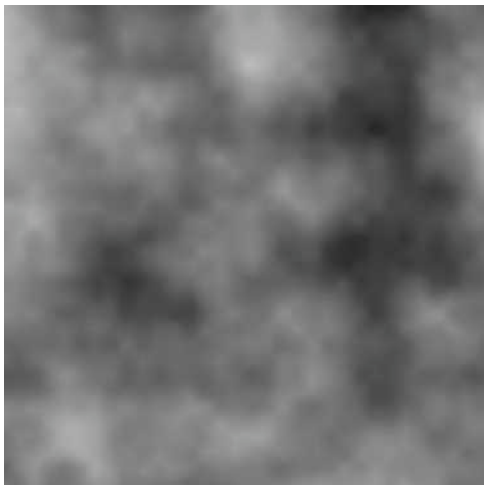
Em comparação com os experimentos realizados na segunda fase, os resultados adquiridos com a combinação *Wander* e *Seek* apresentaram menor quantidade de sedimentos acumulados nos limites do plano. Como esperado, considerando os dados adquiridos ao longo do trabalho (ilustrados nas Figuras 28, 29 e 30), a variação de alturas apresentada com os modificadores com menor valor se demonstraram mais promissoras para a criação de um terreno.

Para analisar os experimentos por uma ótica diferente, pode-se comparar os ruídos formados pelas alternativas dessa fase com uma representação do Ruído de Perlin, utilizado pela ferramenta *ThreeX.Terrain*. A Figura 31 ilustra a visualização de um mapa de altura gerado a partir de Ruído de Perlin. As áreas destacadas em branco representam os valores mais altos do mapa de alturas, e em preto os valores mais baixos.

As Figuras 32 e 33 mostram os mapas de alturas gerados a partir da técnica descrita anteriormente. Esta representação, porém, não apresenta uma técnica de suavização para a geração procedural. Logo, os trajetos percorridos pelos agentes de *steering behaviors* são perceptíveis nas Figuras 32 e 33. Porém, a falta de um mecanismo de suavização não impede a interpretação dos ruídos.

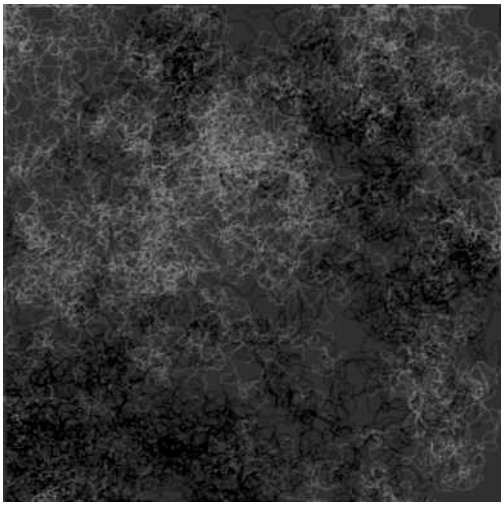
Os três ruídos manifestam suas similaridades, com as áreas em branco com pontos mais concentrados e, ao se aproximar das áreas mais baixas em preto, se encontram mais dispersos.

Figura 31 – Geração de mapa de altura de Ruído de Perlin apresentado na extensão *ThreeX.Terrain*



Fonte – Extensão ThreeX.Terrain

Figura 32 – Ruído resultante do experimento com *Wander* e modificadores de 0.01 e -0.05.



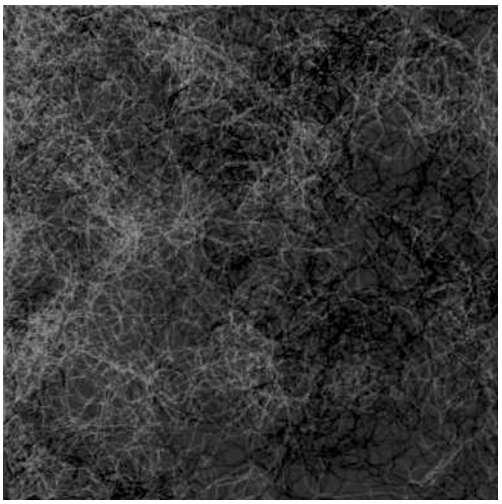
Fonte – Autora

Acredita-se que com o emprego de uma mecanismo de suavização, os ruídos representantes do mapa de alturas criado pelos agentes de *steering behaviors* continuem a demonstrar potencial de utilização considerando a geração procedural de terrenos.

7.4 CONSIDERAÇÕES

Durante as três fases dos experimentos do trabalho, algumas características foram destacadas e podem ser melhor exploradas em trabalhos futuros. Dentre estas características, encontraram-se questões como a concentração de sedimentos nos limites do mapa, consequência da movimentação das entidades *Wander* que ocasionalmente "ficavam presas"na tentativa de se mover para frente.

Figura 33 – Ruído resultante do experimento com *Wander* e *Seek*, e modificadores de 0.01 e -0.05.



Fonte – Autora

Presumiram-se algumas soluções para esse caso em específico. Uma solução consiste na criação de um mapa dinâmico, que expandiria uma vez que uma entidade de *steering behavior* Wander atingisse os limites do mapa de alturas. Uma outra solução mais simples seria a exclusão dos primeiros pontos presentes nos limites do mapa. Ambas as soluções podem não ser ideais, custosas e até comprometer a formação dos ruídos. Elas requerem um estudo detalhado de suas caraterísticas, benefícios e malefícios.

Além disso, ao observar o produto dos experimentos, é inegável a necessidade de um mecanismo de suavização para a conclusão da geração procedural de um terreno. Uma vez que este trabalho propõe e aplica *steering behaviors* para a formação de uma mapa de alturas, propõe-se o estudo da aplicação de *steering behaviors* para a criação de um algoritmo de suavização. A título de exemplo, a aplicação pode conter as entidades Wander que depositam sedimentos no mapa de alturas com o papel de líderes, existindo para cada uma delas um grupo de entidades com o papel de seguidores. Os seguidores, acompanhando a mesma lógica, calculariam a média de alturas dos pontos ao seu redor e conforme o resultado adicionariam ou removeriam sedimentos.

É importante salientar que todos os resultados apresentados nesse trabalho podem ser acessados no repositório *ThreeSteerTerrain*[1]. Os resultados foram divididos em arquivos de imagem e csv com os dados do mapa de alturas. Como padrão de nomenclatura, definiu-se os nomes dos *steering behaviors*, seguidos do número de iterações totais, e por último o valor do iterador de sedimentos. Por exemplo, *Wander-Seek-10-Iterations-20000-heightIterator-0.05.png*. Desse modo, os dados podem ser facilmente acessados e utilizados para análises futuras.

[1] Disponível em `https://github.com/estelavilasboas/ThreeSteerTerrain/`

8 CONCLUSÃO

Este trabalho teve como objetivo averiguar a utilização de agentes de software baseados em *steering behaviors* na construção de um mapa de alturas considerando a geração procedural de terrenos, para seu possível emprego como conteúdos para jogos digitais. Uma vez que a construção de cenários exploráveis é um elemento de impacto significativo na experiência dos jogadores, essa pesquisa se faz necessária para atestar a eficácia de agentes *steering behaviors* para este propósito.

Foram realizados experimentos com as bibliotecas *Three.js*, *ThreeSteer* e a extensão *ThreeX.Terrain*, os quais foram divididos em três fases para facilitar a organização. Cada fase teve um propósito distinto, bem como configurações, resultados e conclusões distintas. Com a execução desses experimentos, mostrou-se ser possível controlar efetivamente o comportamento dos terrenos gerados de acordo com os *steering behaviors* e parâmetros utilizados.

Mantendo a aleatoriedade intrínseca dos mecanismos de geração procedural, os resultados da execução de agentes do *steering behavior Wander* e da combinação entre *Wander* e *Seek* apresentaram uma boa variedade de alturas, podendo ser capazes de criar um terreno com áreas montanhosas e planas dependendo dos valores dos modificadores do mapa de alturas. Essa variedade é muito importante, pois corrobora com a ideia de que os conteúdos de um jogo devem ter originalidade.

Embora não tenha sido elaborada a criação completa de um terreno em si, os ruídos gerados pelos experimentos apresentam boa aleatoriedade e possuem características bem semelhantes às que o Ruído de Perlin apresenta. A diferença mais visível entre as implementações é o fato de que os ruídos gerados pelos experimentos não são suavizados, visto que os agentes apenas incrementavam e decrementavam a altura do exato ponto em que passavam.

Sendo assim, instiga-se sobre a possibilidade de aplicar um mecanismo de suavização (*smoothing*) utilizando *steering behaviors* e uma possível ampliação do campo de pesquisa. A partir de mais estudos sobre a utilização desses agentes para a geração procedural de terrenos, acredita-se que possa ser realizada a criação de uma nova técnica e regras que entreguem resultados originais, estruturados, controláveis, ágeis e atrativos.

Outra pesquisa que traria ótimos avanços para esta área seria categorizar determinados conjuntos de configurações para os *steering behaviors* de acordo com os tipos de conteúdos que eles geram. Essa categorização poderia ser similar ao que Doran; Parberry (8) fizeram, definindo conjuntos de configurações para *coastline*, *smoothing*, *beach*, *mountains* e *rivers*. Outra opção, seria até mesmo gerar macro-categorias por biomas, dependendo do contexto da aplicação.

Também seria válida a criação de uma biblioteca que utilizasse todos estes conceitos simultaneamente para criar um terreno complexo, com diferentes áreas e múltiplos *steering behaviors* trabalhando juntos. Na realidade, os resultados dessa pesquisa não se aplicam exclusivamente para terrenos, pois ruídos são úteis para a criação de diferentes tipos de conteúdo.

Portanto, múltiplos trabalhos poderiam abordar o uso de *steering behaviors* na geração de outros tipos de conteúdo, podendo utilizar este trabalho como base.

Finalmente, pode-se concluir que os objetivos almejados por este trabalho foram alcançados com sucesso, visto que os resultados apresentados dão luz a diversas conclusões significativas para a área. Ainda há muito a ser explorado sobre o assunto, mas a eficácia dessa técnica foi atestada considerando tudo o que foi apresentado.

REFERÊNCIAS

1 AMANDI, A. A. **Programação de agentes orientada a objetivos. Porto Alegre: CPGCC da UFRGS**. 1997. Tese (Doutorado).

2 BEVILACQUA, F.; POZZER, C. T.; D'ORNELLAS, M. C. Charack: Tool for Real-Time Generation of Pseudo-Infinite Virtual Worlds for 3D Games. In: 2009 VIII Brazilian Symposium on Games and Digital Entertainment. [S.l.: s.n.], 2009. p. 111–120. DOI: 10.1109/SBGAMES.2009.21.

3 BEVILACQUA, Fernando. **Understanding Steering Behaviors**. Out. 2012. Disponível em: <https://gamedevelopment.tutsplus.com/series/understanding-steering-behaviors--gamedev-12732>.

4 BUCKLAND, Mat. **Programming game AI by example**. [S.l.]: Jones & Bartlett Learning, 2005.

5 CORREA FILHO, M. A. **Arquitetura de diálogos entre agentes cognitivos distribuıdos. Rio de Janeiro: COPPE da UFRJ**. 1994. Tese (Doutorado).

6 DE CARLI, D. M. et al. Procedural Generation of 3D Canyons. In: 2014 27th SIBGRAPI Conference on Graphics, Patterns and Images. [S.l.: s.n.], 2014. p. 103–110. DOI: 10.1109/SIBGRAPI.2014.41.

7 DE CARLI, Daniel Michelon et al. A survey of procedural content generation techniques suitable to game development. In: 2011 Brazilian Symposium on Games and Digital Entertainment. [S.l.: s.n.], nov. 2011. p. 26–35. DOI: 10.1109/SBGAMES.2011.15.

8 DORAN, Jonathon; PARBERRY, Ian. Controlled Procedural Terrain Generation Using Software Agents. **IEEE Transactions On Computational Intelligence And AI In Games**, v. 2, n. 2, p. 111–119, jun. 2010. DOI: 10.1109/TCIAIG.2010.2049020.

9 GALIN, Eric et al. A Review of Digital Terrain Modeling. **Computer Graphics forum**, v. 38, n. 2, p. 553–577, mai. 2019. DOI: 10.1111/cgf.13657.

10 GREUTER, Stefan et al. Real-time procedural generation ofpseudo infinite'cities. In: PROCEEDINGS of the 1st international conference on Computer graphics and interactive techniques in Australasia and South East Asia. [S.l.: s.n.], 2003. 87–ff.

11 HENDRIKX, Mark et al. Procedural Content Generation for Games: A Survey. **ACM Transactions on Multimedia Computing, Communications, and Applications**, v. 9, n. 1, fev. 2013. DOI: 10.1145/2422956.2422957.

12 JENNINGS, Nicholas R. Coordination techniques for distributed artificial intelligence. John Wiley Sons, 1996.

13 JUCHEM, Murilo; BASTOS, Ricardo Melo. Engenharia de sistemas multiagentes: uma investigação sobre o estado da arte. **Technical Report Series**, PUCRS, n. 14, abr. 2001. Disponível em: <https://www.pucrs.br/facin-prov/wp-content/uploads/sites/19/2016/03/tr014.pdf>.

14 KELLY, George; MCCABE, Hugh. A survey of procedural techniques for city generation. **ITB Journal**, Citeseer, v. 14, n. 3, p. 342–351, 2006.

15 LECHNER, Thomas et al. Procedural modeling of urban land use. In: ACM SIGGRAPH 2006 Research posters. [S.l.: s.n.], 2006. 135–es.

16 OATES, Briony J. **Researching information systems and computing**. [S.l.]: Sage, 2005.

17 OLIVEIRA, Nathan; SEABRA, Rodrigo Duarte. Towards a comprehensive classification for procedural content generation techniques. **XV Simpósio Brasileiro de Jogos e Entretenimento Digital-SBGames**, 2016.

18 POZZER, Cesar Tadeu; FURTADO, Antonio Luz; CIARLINI, Angelo Ernani Maia. **Agentes e emoções em histórias interativas**. [S.l.]: PUC, 2003.

19 REYNOLDS, Craig W. Steering behaviors for autonomous characters. In: CITESEER. GAME developers conference. [S.l.: s.n.], 1999. v. 1999, p. 763–782.

20 ROBSON, Colin; MCCARTAN, Kieran. **Real world research**. [S.l.]: John Wiley e Sons, 2016.

21 SHOHAM, Yoav. Agent-oriented programming. **Artificial intelligence**, Elsevier, v. 60, n. 1, p. 51–92, 1993.

22 SHORT, Tanya X.; ADAMS, Tarn. **Procedural Generation in Game Design**. [S.l.]: CRC Press, 2007.

23 SMELIK, Ruben M.; KRAKER, Klaas Jan de; GROENEWEGEN, Saskia A. A Survey of Procedural Methods for Terrain Modelling. **CASA Workshop on 3D Advanced Media In Gaming And Simulation (3AMIGAS)**, p. 25–34, 2009.

24 SMELIK, Ruben M. et al. A Survey on Procedural Modelling for Virtual Worlds. **Computer Graphics forum**, v. 33, n. 6, p. 31–50, set. 2014. DOI: 10.1111/cgf.12276.

25 TOGELIUS, Julian et al. Search-Based Procedural Content Generation: A Taxonomy and Survey. **IEEE Transactions on Computational Intelligence and AI in Games**, IEEE, v. 3, n. 3, p. 172–186, abr. 2011. DOI: 10.1109/TCIAIG.2011.2148116.

26 WEISS, Gerhard. **Multiagent systems: a modern approach to distributed artificial intelligence**. [S.l.]: MIT press, 1999.

27 WOOLDRIDGE, M. Intelligent agents. In: WEISS, Gerhard. **Multiagent systems: a modern approach to distributed artificial intelligence**. [S.l.]: MIT press, 1999. p. 27–77.

28 WOOLDRIDGE, Michael J; JENNINGS, Nicholas R. Intelligent agents: Theory and practice. **The knowledge engineering review**, v. 10, n. 2, p. 115–152, 1995. Disponível em: <https://eprints.soton.ac.uk/252102/2/publisher_download.pdf>.

29 ZHOU, Howard et al. Terrain synthesis from digital elevation models. **IEEE transactions on visualization and computer graphics**, IEEE, v. 13, n. 4, p. 834–848, 2007.