PONTIFÍCIA UNIVERSIDADE CATÓLICA DO RIO DE JANEIRO

Geração Procedural de pistas de corrida e Inteligência Artificial para o controle de veículos em Jogos Digitais

Gabriel Luiz Neves de Vasconcellos

Projeto Final de Graduação

Centro Técnico Científico - CTC
Departamento de Informática
Curso de Graduação em Engenharia da Computação

Rio de Janeiro, novembro de 2022

Gabriel Luiz Neves de Vasconcellos

Geração Procedural de pistas de corrida e Inteligência Artificial para o controle de veículos em Jogos Digitais

Relatório de Projeto Final, apresentado ao Curso Engenharia da Computação da PUC-Rio como requisito parcial para a obtenção do titulo de Engenheiro de Computação.

Orientador: Augusto Cesar Espíndola Baffa

Rio de Janeiro,
novembro de 2022

Resumo

Vasconcellos, Gabriel Luiz. Baffa, Augusto. Geração Procedural de Pistas de Corrida para Jogos Digitais. Rio de Janeiro, 2022. 33 p. Relatório Final de Projeto Final – Centro Técnico Científico, Departamento de Informática. Pontifícia Universidade Católica do Rio de Janeiro.

A Geração Procedural de Conteúdo é responsável pela automatização da geração de conteúdo com o uso de algoritmos. Nesse projeto, um método para gerar pistas de corrida foi desenvolvido com o objetivo de dispensar a necessidade de planejar pistas de corrida para um jogo digital e garantindo um alto grau de diversidade de pistas que é um dos critérios responsáveis pelo divertimento de um jogador. Outro assunto abordado é a inteligência artificial voltada para o controle de veículos em jogos, com o objetivo de percorrer pistas geradas proceduralmente e validá-las.

Palavras-chave: Geração procedural de conteúdo, inteligência artificial, pistas de corrida, jogos digitais

Abstract

Vasconcellos, Gabriel Luiz. Baffa, Augusto. Procedural Generation of Racing Tracks for Games. Rio de Janeiro, 2022. 33 p. Relatório Final de Projeto Final – Centro Técnico Científico, Departamento de Informática. Pontifícia Universidade Católica do Rio de Janeiro.

The Procedural Generation of Content is used for automating generation of content with algorithms. In this project, a method to generate racing tracks was developed in order to dismiss the necessity of designing racing tracks for a game and ensuring a high degree of diversity that is one of the main criteria responsible for a player's fun amount. Another subject addressed is the artificial intelligence used to control vehicles, with the purpose of travelling along the procedurally generated racetracks and validating them.

Keywords: Procedural generation of content, artificial intelligence, racing tracks, games

Sumário

Lista de ilustrações

1 Introdução

1.1 Contextualização

Os jogos digitais estão cada vez mais presentes no dia-a-dia, chegando a cerca de 3 bilhões de jogadores em 2021[1]. Com o objetivo de manter essas pessoas entretidas, a produção de novos conteúdos que apresentem uma certa diversidade relevante é necessário. Essa diversidade pode ser desde um novo mapa, terreno que o jogador habita, como a posição de recompensas, inimigos ou outro atributo que esteja presente no jogo. Portanto, com o intuito de facilitar a produção de conteúdo, evitando o aumento de trabalho manual para a realização de projetos que envolvam uma grande diversidade de materiais para os jogadores ou até possibilitar a produção, em casos que o ser humano não é capaz de produzir o conteúdo manualmente, a geração procedural de conteúdo surgiu.

A Geração Procedural de Conteúdo (em inglês: Procedural Content Generation ou PCG), conforme definido pelo livro *Procedural Content Generation in Games*[2], é um método algorítmico de geração de conteúdo, podendo utilizar ou não o *input* do usuário. Uma das principais aplicações é o uso do algoritmo no desenvolvimento de jogos digitais para trazer variações em instâncias diferentes do jogo, como diferentes terrenos, localizações de NPCs (Non-player characters, personagens não jogáveis) e distribuição de recursos.

A preocupação pela diversidade de conteúdo nos jogos existe desde o início da década de 80, quando a capacidade de armazenamento era muito limitada e, consequentemente, não permitia a presença de muitos materiais pré-planejados manualmente[3]. O jogo *Rogue* (1980), que aplica o conceito na geração dos calabouços nos quais o jogador explora durante cada partida, é uma das principais influências nesse ramo, dando origem ao gênero *roguelike*.

O *roguelike* é um gênero caracterizado pela geração procedural e muito presente no desenvolvimento de jogos independentes justamente pela redução do custo manual para seus jogos trazido pela automatização feita pelo algoritmo para criar novos níveis[4]. Um exemplo atual desse estilo é o *Hades*[5], ganhador do *The Game Awards* 2020 nas categorias de ação e jogos independentes[6].

O uso da PCG também é presente em jogos fora do gênero *roguelike*, para auxiliar na geração de outros elementos do jogo. A geração de terrenos é um exemplo trazido pelo jogo *Minecraft*[7], no qual toda nova instância do jogo apresenta um mundo gerado proceduralmente com montanhas, árvores, distribuição de recursos, como minérios, e NPCs.

Além dos jogos, a Geração Procedural de Conteúdo é útil para simulações virtuais, como

o gerador de cenários urbanos 3D *ArcGIS CityEngine*[8]. A automatização de conteúdo também é visada na indústria cinematográfica, por exemplo, a *Pixar* desenvolveu uma ferramenta chamada *RenderMan*[9] que faz uso de técnicas que envolvem o uso de geração procedural para, por exemplo, a criação de terrenos e efeitos visuais.

A inteligência artificial (IA) é outro tema abordado nesse projeto. Segundo a definição do livro *Artificial Intelligence: A Modern Approach*[10], dos autores Stuart Russell e Peter Novig, a IA é área de estudo de agentes capazes de interpretar fatores do ambiente para realizar suas ações. Por exemplo, os agentes inteligentes podem desempenhar tarefas como reconhecimento da fala e dirigir carros.

Nesse trabalho, o seguimento da IA utilizado é o voltado para jogos digitais, o uso de agentes inteligentes para simular o comportamento de personagens que não são controlados pelo jogador. O objetivo desse tipo de inteligência é possibilitar um comportamento mais responsivo e adaptável dos NPCs, garantindo uma melhor experiência de jogo.

No caso dos jogos de corrida, a inteligência artificial é responsável pelo controle dos demais veículos que competirão com o humano, procurando simular um piloto mais realista e trazer um maior grau de diversão para o jogo. A maior parte dos agentes inteligentes é baseada em algoritmos de *pathfinding*, com o intuito de encontrar a rota mais curta para ser percorrida nas pistas de corrida.

Um algoritmo de *pathfinding* é utilizado nesse projeto para dizer a direção em que o carro deve ir, evitando que o carro saia da pista e procurando aproximar o comportamento de um piloto artificial com o humano. Ao contrário de algoritmos mais tradicionais, que dependem apenas de valores pré-definidos, esse projeto faz uso de um algoritmo chamado *pathfinding* dinâmico, caracterizado pelo uso de sensores de colisão para auxiliar o veículo em curvas mais acentuadas.

1.2 Objetivos

Nesse projeto, a Geração Procedural de Conteúdo foi estudada e um método para gerar pistas de corrida para jogos digitais, além de métodos para a representação e visualização dos circuitos de corrida com o uso de curvas *Catmull-Rom*. A principal motivação de desenvolver um algoritmo para a geração das pistas de corrida é dispensar a necessidade de construir pistas para o jogo, pois essa tarefa será feito pelo algoritmo. Outro fator facilitado pela automatização é o grau de diversidade proporcionado pelo jogo, já que pistas poderão ser geradas para cada nova corrida, contribuindo para a quantidade de diversão sentida pelo jogador[3].

Outro ponto abordado nesse projeto é o uso de Inteligência Artificial (IA) em conjunto com a Geração Procedural de Conteúdo. A inteligência artificial foi desenvolvida com o objetivo de validar a pista e simular um carro percorrendo a mesma. Ou seja, o foco do

projeto não envolve atingir um desempenho elevado, mas servir como demonstração e validação do resultado da geração das pistas de corrida.

O agente inteligente é baseado em um sistema de *waypoints*, pontos de rota nos quais o carro deve passar, aproveitando os pontos já obtidos durante o processo da geração procedural da pista de corrida. Além disso, sensores de colisão foram implementados para diminuir a velocidade do carro antes das curvas, evitando que o veículo saia da pista em curvas mais fechadas.

1.3 Ambiente

O jogo será desenvolvido na plataforma *MonoGame* [11], um *framework open-source* para o desenvolvimento de jogos digitais multi-plataforma na linguagem *C#* e com suporte ao *.NET* [12]. A principal vantagem do *MonoGame* será agilizar a implementação, pois já possui maneiras simplificadas de renderização de texturas e possibilita configurar rotinas de desenho e atualização conforme a taxa de atualização de quadros do computador.

O objetivo do jogo é simular uma corrida, sendo desenvolvido um sistema de colisão, um comportamento simples para carros de corrida que possua uma maneira de controlar o veículo com o uso de uma inteligência artificial e a própria visualização da pista de corrida.

2 Situação Atual

2.1 Geração Procedural de Conteúdo

De acordo com o artigo *Search-Based Procedural Content Generation: A Taxonomy and Survey*[13], que fez um levantamento e classificou as diferentes implementações da Geração Procedural de Conteúdo, existem três principais abordagens: *search-based*, *constructive* (construtivo) e *generate-and-test* (busca por força bruta).

O algoritmo construtivo gera o conteúdo apenas uma vez e precisa assegurar que está correto ou ótimo durante a construção. Portanto, o método precisa aplicar operações que não permitam danificar o conteúdo.

O algoritmo *generate-and-test* é basicamente um método que gera e testa sucessivamente o conteúdo. Quando um possível candidato é gerado, um teste é feito baseado no critério determinado e chega-se a dois possíveis resultados: o teste falha e os conteúdos candidatos são descartados ou o processo continua até gerar um conteúdo ótimo.

O algoritmo de *search-based* é um tipo de busca por força bruta que ao invés de simplesmente descartar ou continuar testando os seus candidatos até chegar ao resultado ótimo, avaliá-os dando notas. A geração de um novo conteúdo candidato depende da nota atribuída ao conteúdo e tem como objetivo ultrapassar essa nota.

2.2 Geração Procedural de Pistas de Corrida

Em relação a Geração Procedural de pistas de corrida, uma abordagem que já foi utilizada é *search-based* presente no artigo *Towards automatic personalised content creation for racing games*[14]. O critério utilizado para a seleção das pistas geradas foi o valor de diversão para um jogador específico, modelado por uma inteligência artificial. A principal vantagem desse método é permitir a criação de uma pista especializada para um estilo de jogador específico, porém não possibilitaria a criação de pistas em tempo real e exige que um mapeamento do comportamento do jogador já tenha sido feito.

Outro ponto importante da geração das pistas de corrida é maneira em que são armazenadas e representadas. Atualmente, existem duas principais abordagens: pontos de controle ou sequência de segmentos, conforme descrito no artigo *Search-based Procedural Content Generation for Race Tracks in Video Games*[15].

A representação de uma pista de corrida por uma sequência de segmentos permite um maior controle da variedade e curvatura, porém necessitam de uma maior complexidade na sua geração tendo em vista o número de parâmetro necessários para formar um segmento e que sejam possíveis de serem unidos. Enquanto os pontos de controles

possuem apenas a posição como parâmetro e podem ser ligados facilmente com um algoritmo de curvas como Bezier ou Catmull-Rom.

No artigo *Towards automatic personalised content creation for racing games*[14], foram avaliados três métodos para a criação de uma pista representada por pontos de controle: *straightforward*, *random walk* e radial. No *straightforward*, primeiro é formado um retângulo arredondado e perturbações são feitas em seus pontos com base em uma distribuição normal, enquanto o *random walk* realiza essas modificações de forma aleatória. Por fim, o método radial consiste em encontrar pontos pertencentes a uma circunferência e alterar a distância entre os pontos e o centro.

Um exemplo de pistas de corrida feitas por uma sequência de segmentos é o artigo *Search-based Procedural Content Generation for Race Tracks in Video Games*[15]. Os segmentos podem ser retos ou curvos, os retos possuem apenas dois parâmetros: o tipo e comprimento, enquanto os curvos possuem cinco parâmetros: tipo, comprimento, arco da curva, raio A e raio B. Além disso, durante a geração de pistas é utilizado uma função *fitness*, responsável por garantir que os valores cheguem no objetivo desejado, para assegurar que os segmentos formem um circuito, uma pista contínua com os pontos final e inicial iguais, que não haja interseções entre os segmentos da pista e um valor de divertimento mínimo seja alcançado. O valor de divertimento leva em consideração a diversidade da pista com base na sua curvatura, determinada pela quantidade, distribuição e diversidade das curvas, e o perfil de velocidade, determinado pela diversidade presente entre a velocidade no ponto final de cada segmento.

A geração de pistas é feita através de uma população inicial aleatória de segmentos, cada um possuindo sua própria função *fitness*. Após a geração inicial, os parâmetros que definem cada segmento sofrem perturbações e a cada iteração a função *fitness* é aplicada. Caso a condição de parada para a pista seja alcançada, o processo é finalizado e, se o número de iterações exceder o desejado, um novo processo é iniciado com uma nova população inicial.

2.3 Inteligência Artificial para carros

Como foi dito anteriormente nesse documento, a principal maneira de controlar os carros é através do uso de algoritmos de *pathfinding*. Um estudo feito pelo artigo *Game ai: Simulating car racing game by applying pathfinding algorithms* [16] compara algoritmos de *pathfinding* procurando entender qual a melhor abordagem para os carros em jogos de corrida.

Nesse artigo, a pista de corrida possui *waypoints* que são pontos de parada utilizados como nós destinos nos algoritmos implementados e é dividida em nós, representando cada localização possível dentro do cenário do jogo, que são armazenados em um vetor de duas dimensões. Além da posição, os nós também possuem uma cor para representar

obstáculos na pista, os nós com cores brancas representam obstáculos enquanto nós de outras cores representam espaços livres para a passagem do veículo. A partir da definição do vetor que representa os possíveis nós da pista de corrida, foram implementados quatro algoritmos de *pathfinding*: um algoritmo A*, duas modificações do algoritmo A* e um algoritmo de *pathfinding* dinâmico.

O algoritmo A* utiliza quatro nós adjacentes à posição atual do veículo, representando as possibilidades nas seguintes direções: esquerda, direita, frente e trás. Cada um desses nós é pontuado, baseado numa função heurística, representada na equação (2.1) abaixo, que calcula o custo total, *f(n)*, a partir do custo para chegar até o nó a partir da posição inicial, *g(n)*, e de uma estimativa do custo para chegar no destino a partir daquele nó, *h(n)*. A cada iteração do algoritmo, o nó com menor custo total, *f(n)*, é escolhido e o veículo ocupa sua posição, tornando-se a nova posição atual.

$$f(n) = h(n) + g(n) \tag{2.1}$$

As modificações do algoritmo A* consistem em diminuir o número de nós utilizados no cálculo do caminho mais curto. A primeira modificação procura reduzir a quantidade de *waypoints* usando um algoritmo de linha de visão, diminuindo o tempo para calcular o caminho entre cada nó até o destino. O algoritmo de linha de visão verifica a existência de qualquer obstáculo entre o veículo, nó atual, e os próximos *waypoints*, nós destinos, caso não haja um obstáculo entre o *waypoint* mais distante e o carro, os demais são descartados e a rota calculada pelo algoritmo considera apenas o mais distante. Ao observar a figura 1 como exemplo, como o *waypoint* representado pelo número 3 esta mais distante que o número 2 e não possui obstáculo conforme a linha de visão, o *waypoint* 2 seria descartado e o caminho calculado seria de 1 para 3 ao invés de 1 para 2 e depois 2 para 3.

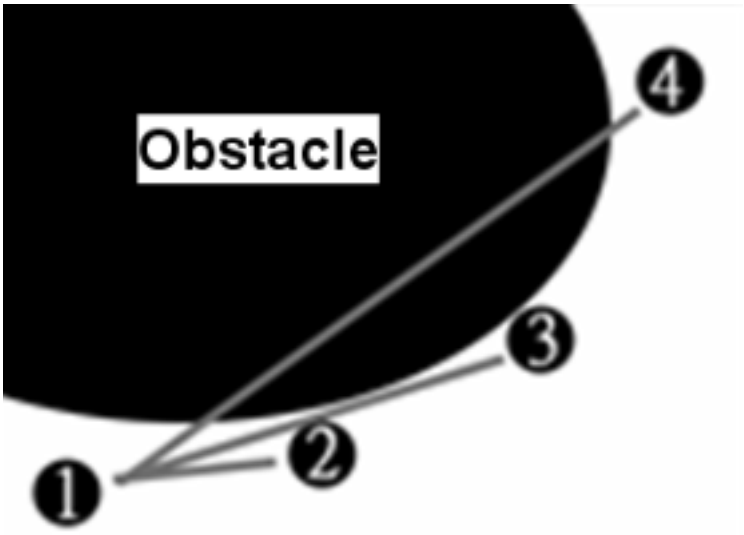

Figura 1 – Algoritmo linha de visão

A segunda modificação do algoritmo A* leva em consideração que o veículo consegue dirigir para frente sozinho. Com a condição citada, os quatro nós utilizados mudam para três nós que ficam em posições relativas a frente do veículo: frente direita, frente central e frente esquerda. Dessa maneira, o número de nós reduz e otimizam o tempo de computação do algoritmo A*.

O algoritmo de *pathfinding* dinâmico faz uso de sensores de colisão posicionados em duas posições do veículo: frente esquerda e frente direita. Os dois sensores possuem a mesma posição no eixo vertical do veículo, configurada por uma variável x, e posições diferentes, distanciadas pela metade da largura do carro denominada y, no eixo horizontal do veículo, conforme ilustrado na figura 2. Durante o percurso, os sensores verificam a cor do nó onde estão posicionados e são acionados quando a cor for correspondente a de um obstáculo, neste caso a cor branca.

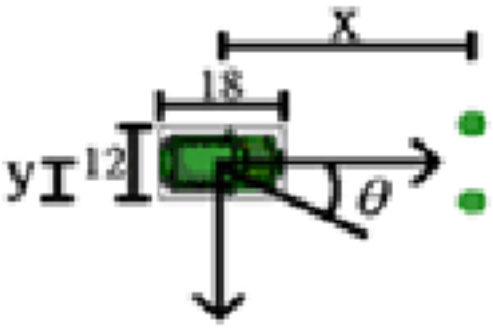

Figura 2 – Sensores de colisão

Quando a colisão é detectada, o carro rotaciona para mudar sua direção frontal de acordo com o sensor acionado: se a colisão ocorre no sensor da direita o carro gira no sentido horário, enquanto na esquerda gira no sentido anti-horário. Como ilustrado na figura 3, caso os dois sensores sejam acionados ao mesmo tempo, outros dois sensores são utilizados para decidir em qual sentido o carro deve rotacionar.

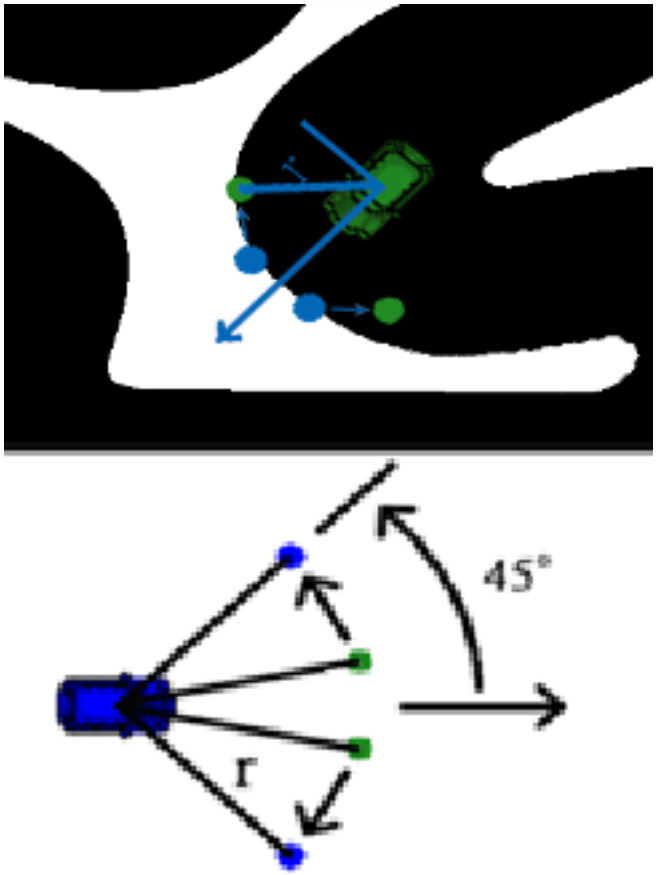

Figura 3 – Demonstração do uso dos sensores extras

Por fim, testes para comparar qual abordagem possui o melhor desempenho foram realizados. O resultado final indicou que a segunda modificação para o algoritmo A*, que utiliza apenas os nós frontais, teve o menor tempo para realizar a volta, porém o algoritmo de *pathfinding* dinâmico mostrou um comportamento mais suave e natural ao percorrer a pista. Outra vantagem do *pathfinding* dinâmico é quando existe obstáculos aleatórios e pode ser feito em tempo real, ao contrário das demais abordagens que necessitam dos *waypoints* e dos nós. A primeira modificação também mostrou resultados positivos, obtendo uma redução em torno de 97% na quantidade de *waypoints* que eram utilizados pelo algoritmo A*.

3 Fundamentos

3.1 Curvas Catmull-Rom

3.1.1 Motivação do uso de curvas *Catmull-Rom*

A representação da pista de corrida escolhida foi o conjunto de pontos de controle que representam um conjunto de curvas *Catmull-Rom*[17], devido ao nível de continuidade e a garantia da passagem da curva por todos os pontos definidos.

A união de curvas *Catmull-Rom* apresenta uma continuidade *C1*, ou seja, duas curvas apresentam a mesma primeira derivada no ponto em que se unem, garantindo uma maior suavidade e ausência de "pontas", conforme ilustrado na figura 4 extraída da apresentação *Hermite and Catmull-Rom Curves*[18]. Portanto, algoritmos limitados a um nível de continuidade *C0*, como a curva *Bezier*[19], ou que não passem por todos os pontos de controles definidos foram descartados para evitar problemas na geração da pista.

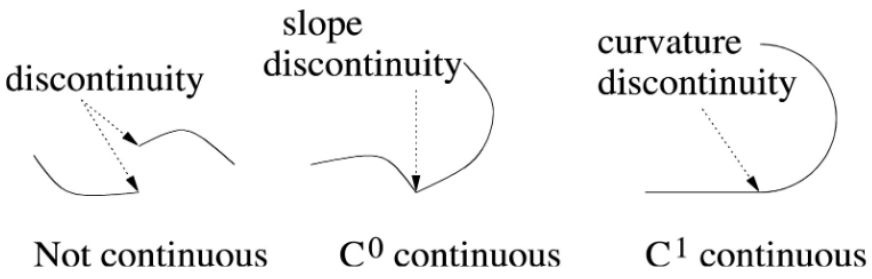

Figura 4 – Níveis de continuidade

3.1.2 Cálculo da curva *Catmull-Rom*

A curva *Catmull-Rom* é um tipo de curva cúbica, definida pelo polinômio de terceiro grau:

$$p(t) = at^3 + bt^2 + ct + d \tag{3.1}$$

Para calcular os coeficientes da equação, quatro pontos de controle são utilizados, sendo p_1 e p_2 os pontos correspondentes ao início e fim da curva e os outros pontos, p_0 e p_3, fazem parte, respectivamente, da curva anterior e posterior à curva que será desenhada. A imagem 5, extraída do artigo *Spline approximation-based data compression for sensor arrays in the wireless hydrologic monitoring system*[20], representa um exemplo de uma curva desenhada pelo método *Catmull-Rom*, na qual p_1 é representado por p_i.

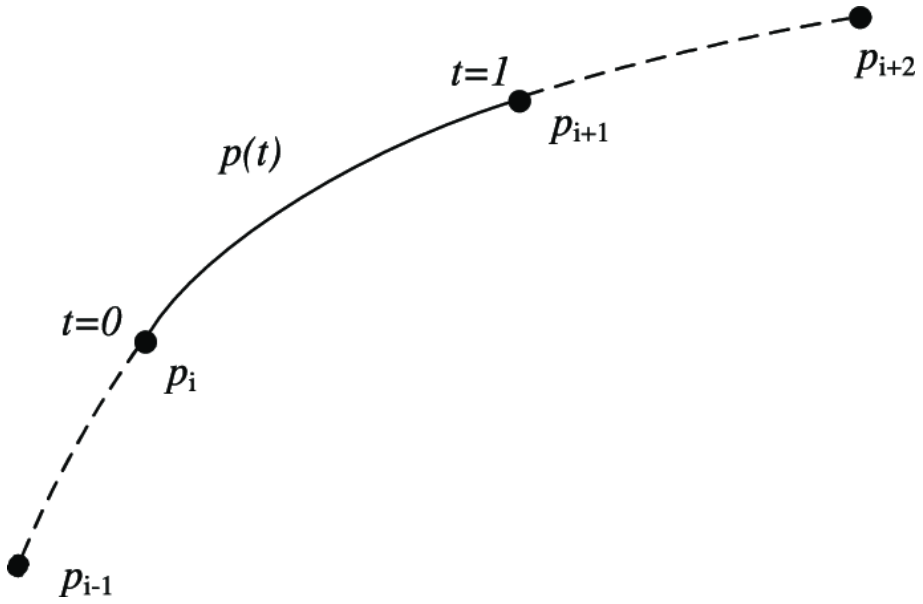

Figura 5 – Método Catmull-Rom

Outro aspecto importante para o cálculo dos coeficientes são as tangentes nos pontos p_1 e p_2, que devem ser iguais tanto para a curva que inicia em p_0 quanto a que termina em p_3, assim garantindo a continuidade *C1*. Com as restrições estabelecidas é possível reformular a equação 4.1 em relação aos quatro pontos citados, sendo *p(t)* um ponto entre p_1 e p_2 e $t \in [0, 1]$:

$$
\begin{aligned}
p(t) &= \frac{1}{2} \cdot (q_0 p_0 + q_1 p_1 + q_2 p_2 + q_3 p_3) \\
q0 &= -t^3 + 2t^2 - t \\
q1 &= 3t^3 - 5t^2 + 2 \\
q2 &= -3t^3 + 4t^2 + t \\
q3 &= t^3 - t^2
\end{aligned}
\tag{3.2}
$$

3.1.3 Tangente e curvas paralelas

Com o intuito de permitir desenhar as curvas paralelas que servem para delimitar as pistas de corrida, ilustradas na figura 6 com a cor branca, é necessário calcular o vetor normal da curva Catmull-Rom em cada um de seus pontos. Para calcular esse vetor, é feito o uso da tangente da curva, que pode ser obtida pela primeira derivada da equação 3.2, resultando na equação 3.3.

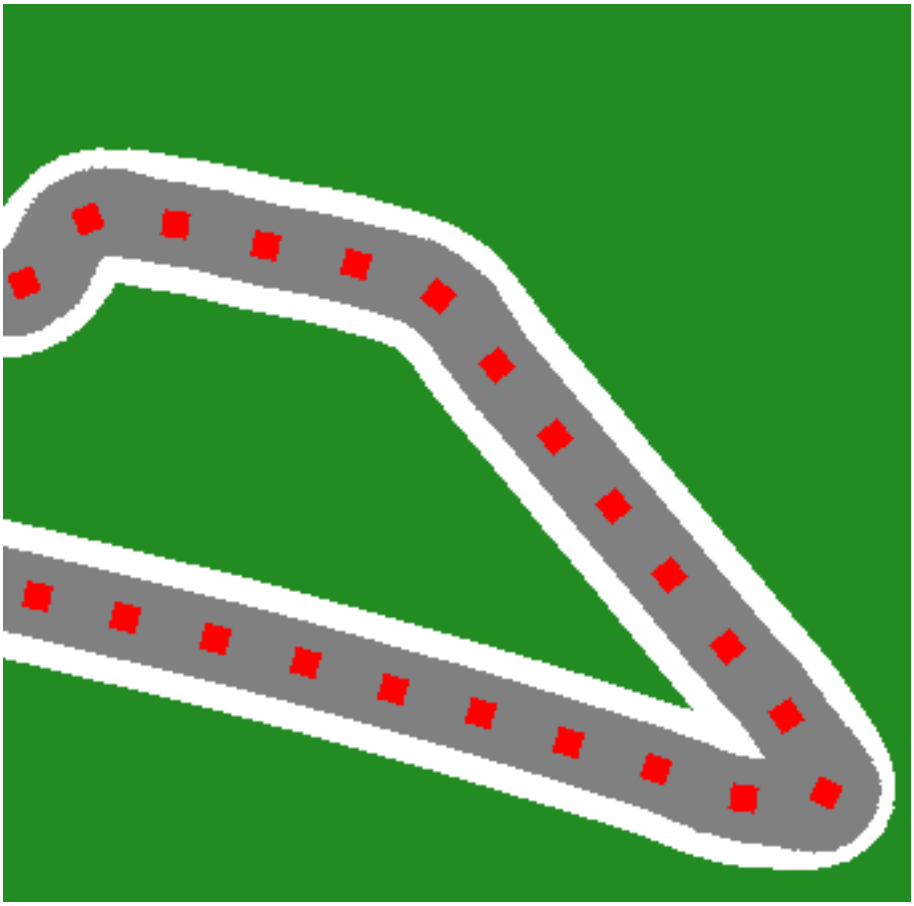

Figura 6 – Curvas paralelas em branco na pista de corrida

$$
\begin{aligned}
p(t) &= \frac{1}{2} \cdot (q_0 p_0 + q_1 p_1 + q_2 p_2 + q_3 p_3) \\
q0 &= -3t^2 + 4t - 1 \\
q1 &= 9t^2 - 10t \\
q2 &= -9t^2 + 8t + 1 \\
q3 &= 3t^2 - 2t
\end{aligned}
\tag{3.3}
$$

Após obter a tangente no ponto desejado, é possível calcular um vetor perpendicular a ela, aplicando uma matriz de rotação, demonstrado na equação 3.4, em relação ao ângulo reto. No caso bidimensional, dado uma tangente definida pelo vetor *(a, b)* e utilizando a matriz de rotação, obtém-se o vetor perpendicular *(-b, a)*. Para obter o vetor normal a partir do vetor perpendicular, basta normalizá-lo, conforme a equação 3.5.

$$R(\theta) = \begin{bmatrix} cos(\theta) & -sin(\theta) \\ sin(\theta) & cos(\theta) \end{bmatrix} \tag{3.4}$$

$$\hat{u} = \frac{\vec{u}}{\|\vec{u}\|} \tag{3.5}$$

Com o vetor normal obtido, cada ponto de uma curva paralela pode ser calculada somando o vetor normal multiplicado por um fator d, responsável pela distância entre a curva central e a paralela, ao ponto desejado, conforme demonstrado na figura 7. No caso das pistas desenhadas nesse projeto, o fator d é sempre a metade do tamanho da pista de corrida para uma das curvas e para a outra é o mesmo valor multiplicado por -1, com o objetivo de criar uma curva paralela na direção oposta.

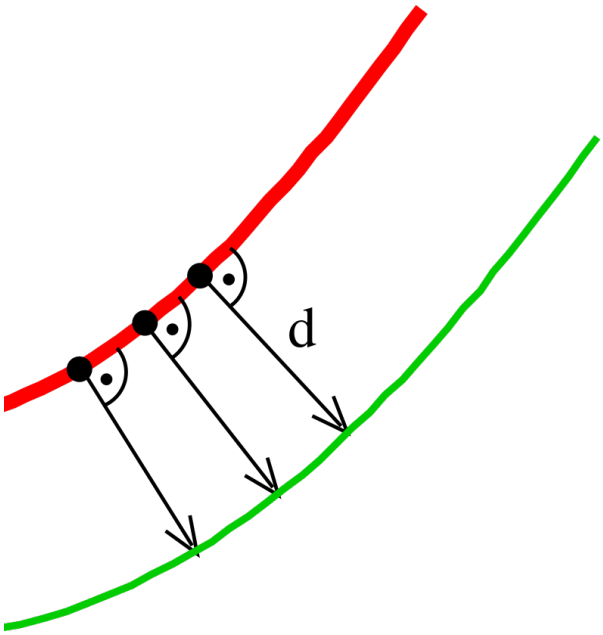

Figura 7 – Curva paralela

3.2 Funcionamento do MonoGame

O *MonoGame*[11] foi escolhido para auxiliar no desenvolvimento de um jogo capaz de simular uma corrida em uma pista. A ferramenta funciona com o uso de uma classe principal chamada *Game*, que possui cinco métodos principais definidos: *Initialize*, *LoadContent*, *Update*, *Draw* e *UnloadContent*, que são executados conforme ilustrado na figura 8.

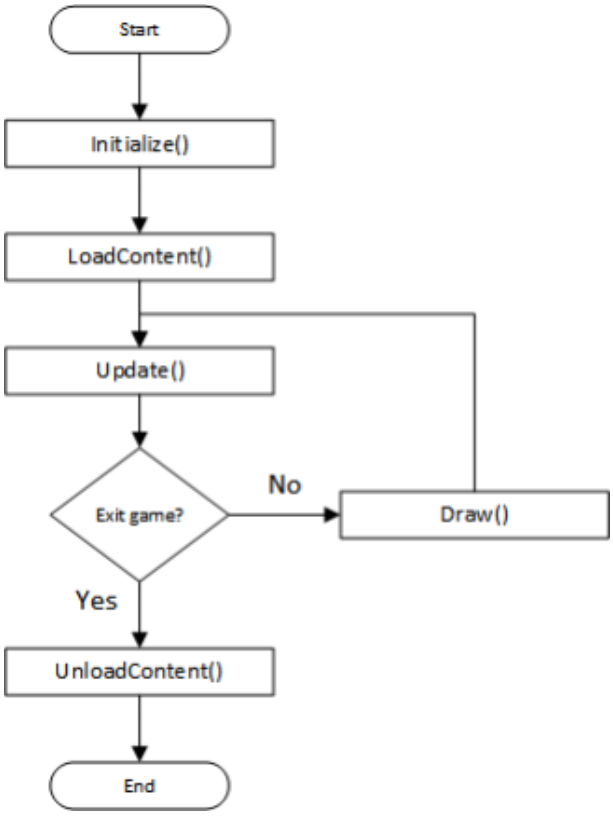

Figura 8 – Funcionamento do Monogame

O método *Initialize* é executado assim que o jogo inicia, antes da primeira atualização e renderização do estado atual do jogo, onde ocorre a execução dos métodos *Update* e *Draw*. A principal funcionalidade do método é definir as configurações necessárias para o primeiro estado do jogo, como a criação dos objetos que participam da cena do jogo.

Enquanto o *Initialize* é executado, o processo definido no método *LoadContent* é iniciado. Essa função é responsável pelo carregamento e configuração de recursos gráficos do jogo, como imagens e texturas. Dentro desse método, também é inicializado uma instância da classe *SpriteBatch* que é utilizada para o desenho de texturas do jogo e, consequentemente, precisa configurar e armazenar recursos gráficos.

Após a execução dos métodos *Initialize* e *LoadContent*, o jogo inicia seu ciclo de atualização e renderização, na qual os métodos *Update* e *Draw* são chamados sucessivamente. O método *Update* realiza a atualização do estado atual do jogo, por exemplo, funções que checam colisões e utilizam o *input* do jogador devem ser chamados nesse momento. Quando o *Update* é finalizado, o *Draw* é chamado para renderizar o estado atual do jogo, podendo conter chamadas de métodos do *SpriteBatch* para desenhar elementos gráficos na tela.

Por fim, ao chegar uma solicitação para o término do jogo, o método *UnloadContent* é chamado. Esse método é responsável por liberar os recursos usados durante o

carregamento de objetos, como textura e imagens, feitos pelo método *LoadContent*.

4 Proposta

4.1 Configuração do ambiente

Como recomendado pela própria documentação do *MonoGame*[21], foi instalado e configurado o *Microsoft Visual Studio 2022*[22]. A vantagem dessa *IDE* é a integração com o sistema *.NET* que será utilizado para executar o jogo desenvolvido para o *MonoGame*.

4.2 Atualização e renderização de objetos do jogo

Dentro da classe *Game*, explicada em detalhes na seção 3.2, existem duas rotinas essenciais para atualizar o estado atual do jogo e renderizá-lo: *Update* e *Draw*. E como cada objeto do jogo possui seu próprio estado, que precisa ser atualizado e renderizado durante a execução do jogo, foi implementada uma classe *GameObject* para facilitar o desenvolvimento de cada novo elemento do jogo e automatizando chamadas para rotinas de atualização e renderização.

A classe *GameObject* é herdada por todos os objetos do jogo e obriga a implementação dos métodos *Update* e *Draw*. Além dos métodos citados, possui o *LoadContent* que pode ou não ser implementado, já que nem sempre é necessário carregar um recurso gráfico como imagens para cada objeto. Ao inicializar uma instância dessa classe, ela é adicionada em um vetor *AllGameObjects*, contendo todas as instâncias de *GameObject* e permitindo iterar cada uma delas para executar os seus métodos *Update*, *Draw* e *LoadContent* dentro dos métodos respectivos em uma instância de *Game*.

4.3 Representação da pista de corrida

Primeiramente, para atender o comportamento de uma interpolação de curvas *Catmull-Rom*, foi desenvolvida uma classe chamada *Spline*. A *Spline* contém uma lista com todos os pontos de controle e métodos para calcular a posição e a tangente, obtida através da primeira derivada, nos eixos X e Y correspondente a um ponto em dado momento t da curva.

Sobre o parâmetro t, se for um número inteiro, é obtido a posição do ponto de controle correspondente ao índice passado e, caso seja um número com casas decimais, a posição retornada é relativa ao cálculo do algoritmo *Catmull-Rom* na equação (3.2). Por exemplo, um valor de t igual a *1.5* utilizado como parâmetro da função retornaria uma posição entre os pontos de controle com índices 1 e 2 conforme o algoritmo.

Com o intuito de renderizar a pista de corrida baseada na curva definida pela *Spline*, a classe *RacingTrack*, herdeira do *GameObject*, foi desenvolvida. Em seu método *Draw*,

valores reais entre 0 até a quantidade de pontos de controle são passados como parâmetro t para as funções responsáveis pelo cálculo da posição e primeira derivada presentes na *Spline*. Com os dados obtidos, é possível posicionar retângulos e calcular suas rotações para acompanhar a curva, que são formados pelo ângulos entre as tangentes que são equivalentes as primeiras derivadas obtidas e o eixo Y, compondo a faixa principal da pista de corrida.

Além da faixa principal do circuito, é preciso calcular o posicionamento e rotação de retângulos para compor os limites da pista. Para encontrar a posição, basta seguir os procedimentos descritos na subseção 3.1.3 para encontrar pontos de curvas paralelas, enquanto a rotação, é feita da mesma maneira que os retângulos da faixa principal.

Após o desenho de todos os retângulos, é gerado uma textura final que é renderizada sem precisar refazer os cálculos descritos acima, otimizando o método *Draw* da *RacingTrack*. Sendo assim, todas as computações necessárias não precisam ser feitas novamente depois da primeira renderização, já que todas as informações estão presentes na textura gerada.

Para gerar uma pista teste, foi feito um algoritmo de leitura de um documento no formato *CSV* com as posições dos pontos de controle. O arquivo utilizado é uma representação de um circuito real chamado *Brands Hatch* e foi extraído de um *dataset* do *GitHub*[23]. Segue uma imagem do resultado, na qual a cor cinza representa a faixa principal, vermelho os pontos de controle e branco os limites do circuito:

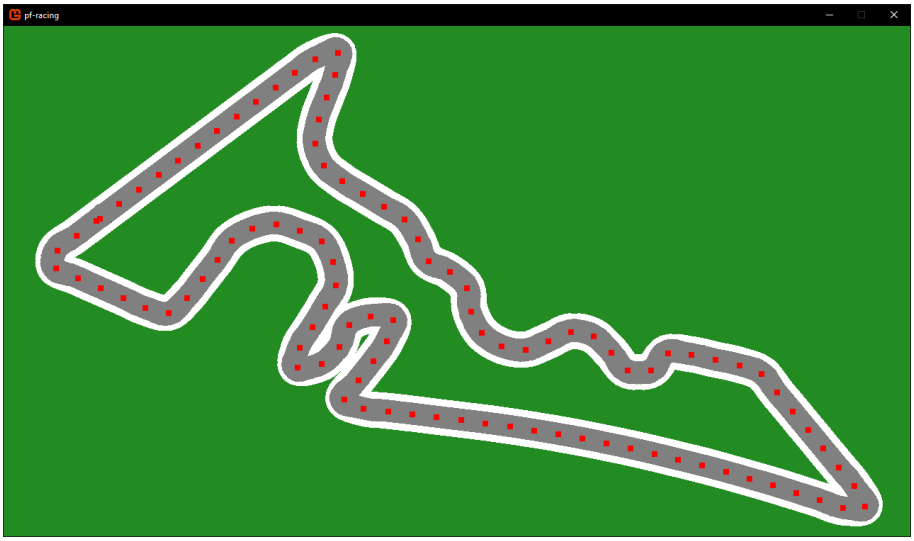

Figura 9 – Circuito Brands Hatch

4.4 Geração da pista de corrida

A abordagem para a geração de pistas é o algoritmo *generate-and-test*. Após a geração de uma pista inicial, uma inteligência artificial garante que o conteúdo é válido, percorrendo a pista do início ao fim. Se nenhum problema ocorrer, a pista é considerada valida e o

processo de termina, caso contrário, outra pista é gerada até um conteúdo valido ser gerado.

Em relação ao método para geração da pista de corrida, uma pista fechada em que o ponto de partida é o mesmo que o de chegada, foi implementado uma versão do método radial, citado na seção 2.2. Os pontos de controle que formam a pista de corrida são gerados percorrendo uma circunferência no sentido anti-horário de x em x ângulos, o valor de x é a razão entre 2π e o número de pontos que configura a pista, obtendo um ângulo θ diferente para cada ponto e posicionando os pontos conforme uma distância variável multiplicada pelo vetor formado pelo seno e cosseno do ângulo θ obtido. Para permitir um maior controle sobre os resultados do método, são disponibilizados 3 parâmetros: um fator de estabilidade, número de pontos e largura da pista.

O fator de estabilidade é responsável pela variação da distância dos pontos de controle do centro da circunferência. O cálculo da distância para um ponto específico é feito da seguinte maneira: soma-se o raio inicial da circunferência multiplicado pelo fator de estabilidade com o valor do raio multiplicado por um número real aleatório. Portanto, quanto maior o valor do coeficiente de estabilidade, mais próximo de uma circunferência ficará a pista. Para um melhor entendimento, as imagens de dois exemplos foram disponibilizadas, ambos formados por 10 pontos e com a mesma largura de pista, porém o primeiro, representado pela figura 10, com o fator de estabilidade igual a 1 e outro, ilustrado na figura 11, com o fator de estabilidade igual a 5.

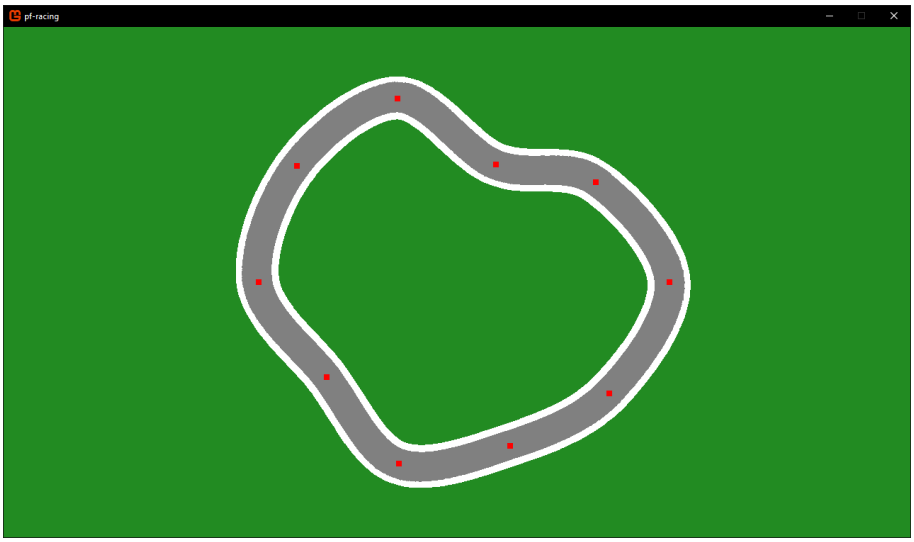

Figura 10 – Pista de corrida gerada com fator de estabilidade 1 e número de pontos igual a 10

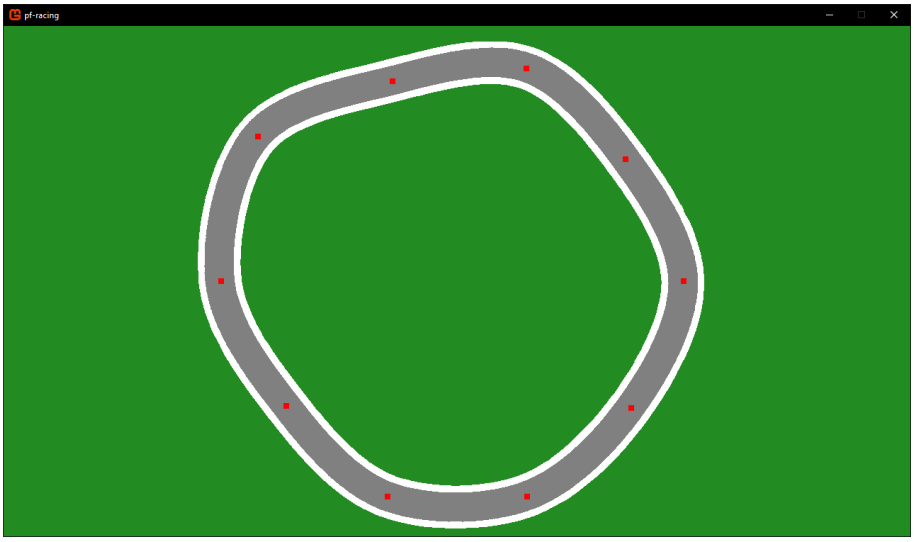

Figura 11 – Pista de corrida gerada com fator de estabilidade 5 e número de pontos igual a 10

Outro parâmetro que afeta diretamente o circuito gerado é o número de pontos. O comportamento observado pelo alto número de pontos é o aumento no número de curvas da pista, podendo se assemelhar a uma flor com fatores de estabilidade mais baixos e com uma circunferência com pequenas deformidades. Duas imagem representativas demonstram os casos com alta e baixa estabilidade, sendo a primeira, figura 12, com o valor de estabilidade 1 e segunda, figura 13, com o fator de estabilidade 5, ambos com o número de pontos igual a 30.

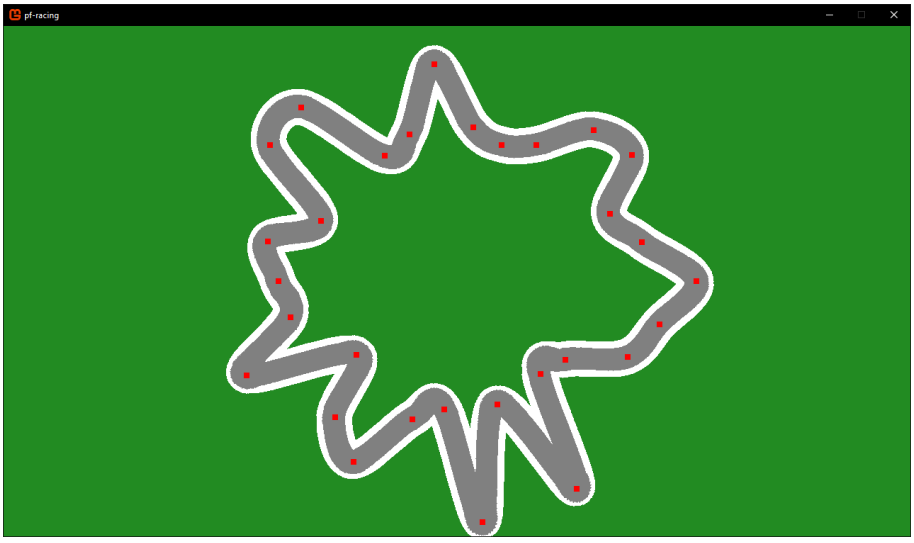

Figura 12 – Pista de corrida gerada com fator de estabilidade 1 e número de pontos igual a 30

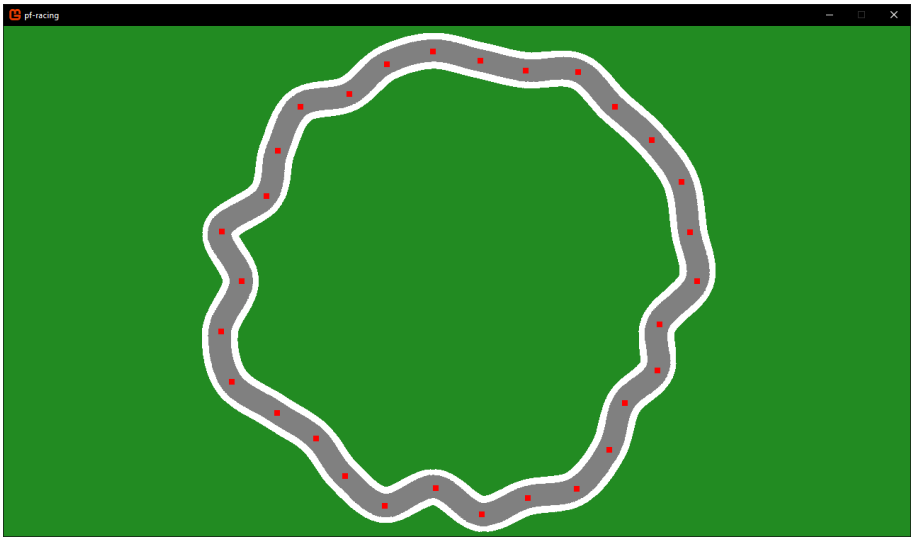

Figura 13 – Pista de corrida gerada com fator de estabilidade 5 e número de pontos igual a 10

4.5 Comportamento do veículo

Para permitir o uso de uma IA, foi realizado uma implementação simplista de um carro não-realista em uma classe chamada *CarEntity*, que também herda da classe *GameObject*. Além das rotinas de desenho e atualização, que são características do *GameObject*, a *CarEntity* implementa checagem de colisão e processamento de possíveis *inputs* feitos por um jogador ou uma inteligência artificial para controlar aspectos da movimentação, como a rotação e velocidade do veículo.

A colisão é realizada pelo mapeamento de pontos de uma circunferência centrada no carro, verificando se a cor da textura é verde, que simboliza a área externa à pista de corrida, na localização de cada ponto. Apenas 10 pontos distribuídos homogeneamente pelo perímetro da circunferência são avaliados, que são suficientes para a finalidade desejada: checar se o veículo ainda está na pista.

O processamento do *input* é feito pela decomposição em dois eixos, X e Y, nas quais ambos variam de -1 a 1. O eixo X é responsável pelas variações na rotação do veículo: valores negativos representam o sentido anti-horário, valores positivos representam o sentido horário e, se o valor for zero, nenhuma rotação ocorre. O eixo Y define o estado de aceleração do carro: se o valor for positivo, o acelerador é ativado, e, no caso de valores negativos ou nulo, o freio é ativado.

Em relação a parte de movimentação, a posição futura do veículo é definida por dois fatores: a direção e a velocidade linear. A direção é definida pelo seno e cosseno da rotação atual do carro, sendo alterada por uma velocidade angular pré-definida que pode ser anulada ou mudar de sentindo conforme o *input* de quem controla o veículo. A velocidade linear não pode ser negativa, ou seja, o carro não pode dar ré e é incrementada

pela aceleração, conforme o *input*: quando o acelerador é ativado, a aceleração é positiva, aumentando a velocidade linear até um limite pré-definido, e, se o freio for ativado, a aceleração é negativa, diminuindo a velocidade linear até zero.

4.6 Implementação da inteligência artificial

O algoritmo responsável pela inteligência artificial é semelhante ao *pathfinding* dinâmico documentado na seção 2.3, porém ao invés do carro apenas utilizar sensores, também são aproveitados *waypoints* presentes na pista de corrida. Em relação a parte dinâmica, o veículo possui apenas um ponto de colisão na frente do veículo que serve para detectar a presença de curvas, enquanto os *waypoints* são usados como nós destinos pelos quais o carro sempre passa, dando uma direção na qual o veículo pode percorrer a pista.

Em relação ao funcionamento da detecção de uma curva, o sensor de colisão é sempre posicionado numa posição futura do veículo, baseado na velocidade linear e direção atual, com o objetivo de prever se permaneceria na pista ao continuar na direção e velocidade que esta navegando. Caso o sensor de colisão seja acionado, significa que uma curva esta chegando e o carro reduz sua velocidade, enviando um *input* com o eixo Y negativo, para não sair do circuito enquanto rotaciona para acompanhar a curva. Se o sensor não está sendo acionado, um *input* com o eixo Y positivo é enviado para sinalizar que o carro deve permanecer acelerando pela ausência de perigo.

Enquanto a velocidade é controlada pelo sensor de colisão, os *waypoints* são responsáveis pela direção do veículo. Durante a corrida, a rotação do veículo é alterada conforme a diferença entre os ângulos formados pelo vetor direção, que sempre aponta para frente em relação ao carro, e o vetor formado entre o carro e o ponto correspondente a localização da próxima parada. Sendo assim, a inteligência artificial envia um *input* no eixo X relativo ao sentido do ângulo, ou seja, valores positivos são enviados para girar no sentido horário e valores negativos são enviados para girar no sentido anti-horário.

4.7 Testes e resultados

Para entender os diferentes perfis que podem ser gerados pela pista, valores para os parâmetros fator de estabilidade e número de pontos do gerador de pistas foram testados. Como a criação da pista depende da aleatoriedade, pistas com o mesmo parâmetro podem ter aparências bem diferentes e quanto menor o fator de estabilidade mais difícil é gerá-la novamente.

A partir do que já foi relatado na seção 4.4, Geração de Pistas de Corrida, foi possível selecionar uma faixa de valores mais apropriada para análise das pistas geradas. Os valores de estabilidade muito altos deixam a pista próxima de uma circunferência e com pouca variedade, como nas figuras 13 e 11, e valores muito baixos causam pistas muito

imprevisíveis, portanto, foram limitados para valores entre 0.5 e 2.

Em relação ao números de pontos, valores elevados geram pistas parecidas com uma flor, como na imagem 12, precisando limitar a 20 pontos por pista para evitar esse comportamento. Além disso, número de pontos inferiores a oito foram descartados, já que geram pistas muita parecidas independente do valor do fator de estabilidade e menos de três pontos resultam em pistas inválidas, como linhas retas e pontos. Por exemplo, grande parte das pistas geradas com sete pontos apresentam pouca diversidade, como mostrado na figura 14 e, ao utilizar apenas três pontos, todas as pistas possuem um formato triangular como na imagem 15.

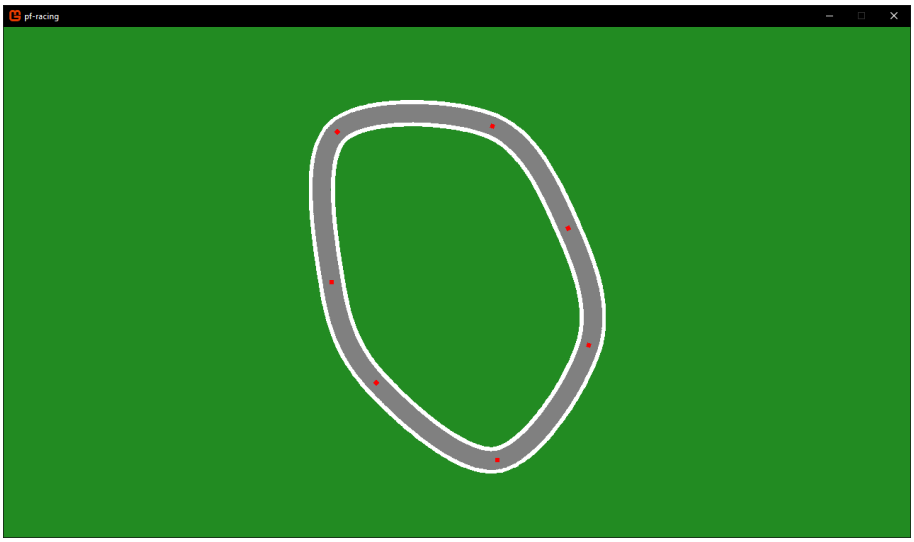

Figura 14 – Pista de corrida gerada com fator de estabilidade 1 e 7 pontos

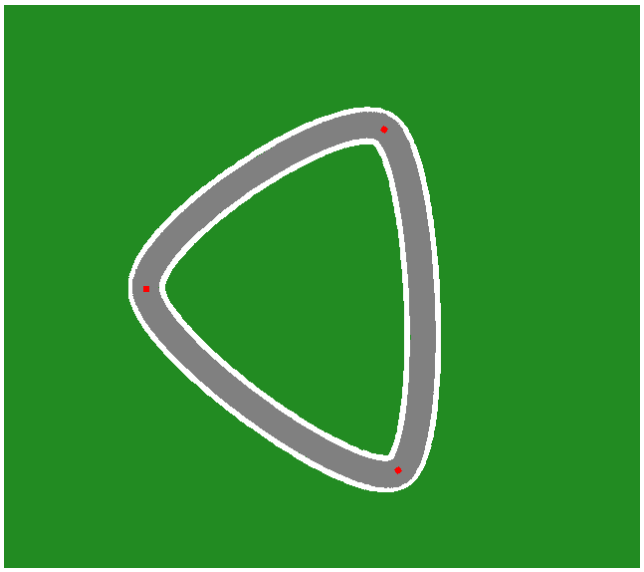

Figura 15 – Pista de corrida triangular gerada com três pontos e fator de estabilidade 1

Após delimitar a faixa de valores para o fator de estabilidade e o número de pontos, os resultados começaram a ficar mais próximos de pistas de corridas reais e foi possível estabelecer valores ideais para cada um dos parâmetros. Como mostrado nas figuras 17 e 18, o número de pontos entre 12 e 20 trouxeram pistas com maior número de curvas, podendo ser utilizado para gerar pistas maior diversidade, e o fator de estabilidade serviu como um seletor de dificuldade, pois em valores mais elevados as curvas tendem a ser menos acentuadas. Em adição, o valor de estabilidade abaixo de 1 aumenta a ocorrência de pistas muito irregulares, como na imagem 16, obtendo melhores resultados com valores entre 1 e 2.

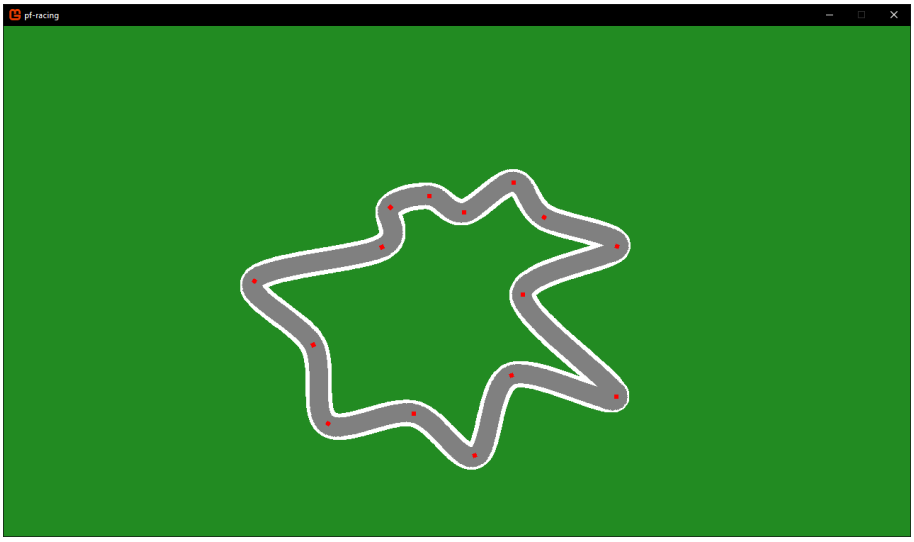

Figura 16 – Pista de corrida gerada com fator de estabilidade 0.5 e 15 pontos

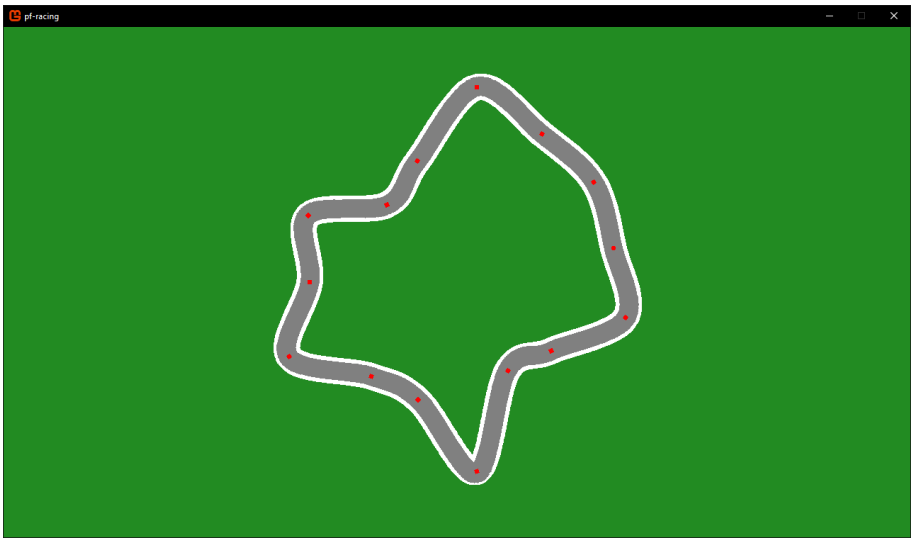

Figura 17 – Pista de corrida gerada com fator de estabilidade 1 e 15 pontos

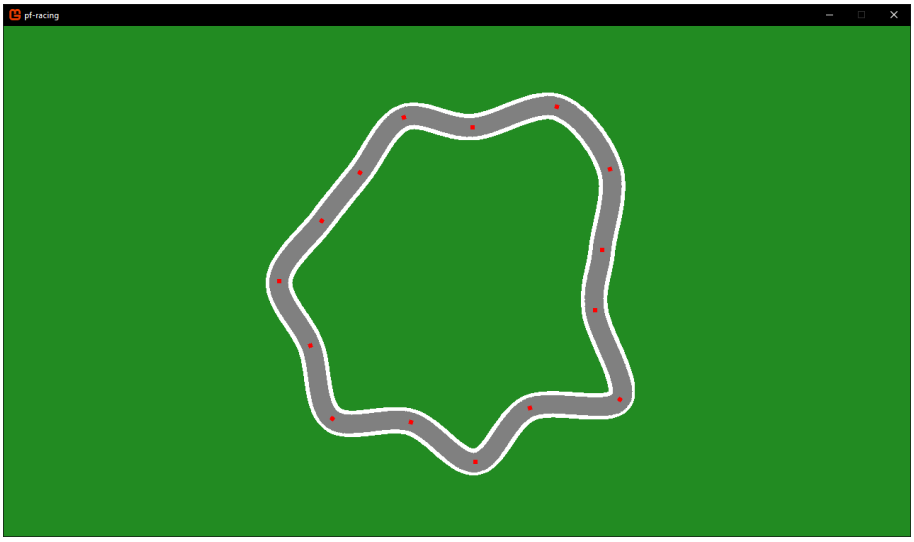

Figura 18 – Pista de corrida gerada com fator de estabilidade 2 e 15 pontos

Ao reduzir o número de pontos para valores de 8 a 11, podemos observar padrões com curvas menos acentuadas e formatos mais regulares. Para pistas formadas por oito e nove pontos, o fator de estabilidade pareceu apresentar melhores resultados entre 0.5 e 1, representados nas imagens 19 e 20, apresentando uma grande variedade. Enquanto, pistas formadas por 10 e 11 pontos apresentaram melhores resultados com o fator de estabilidade entre 0.9 e 1.5, como as figuras 21 e 22, já que valores muito baixos são muito irregulares. Além disso, nessa faixa de número de pontos, o fator de estabilidade acima de 1.5 apresentou um aumento na geração de pistas circulares, semelhantes a figura 14, principalmente nos casos com 10 e 11 pontos.

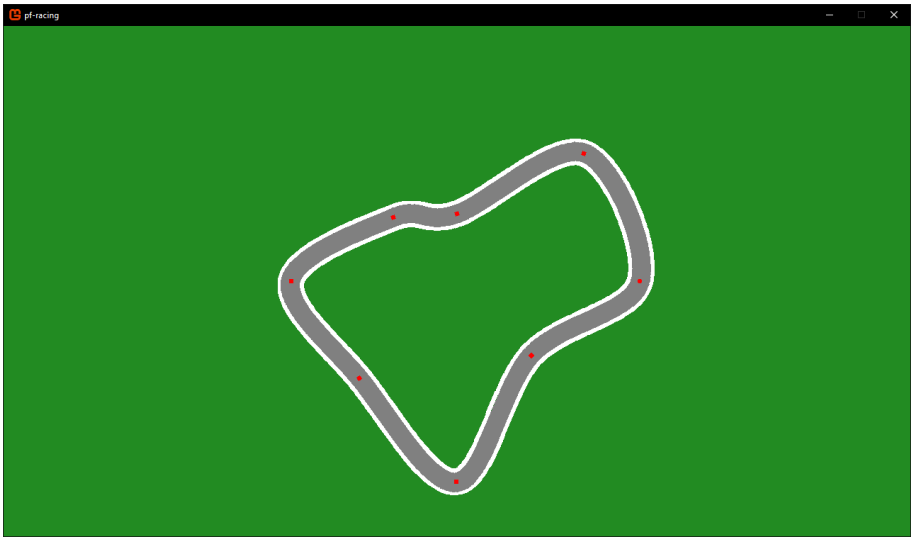

Figura 19 – Pista de corrida gerada com fator de estabilidade 0.5 e 8 pontos

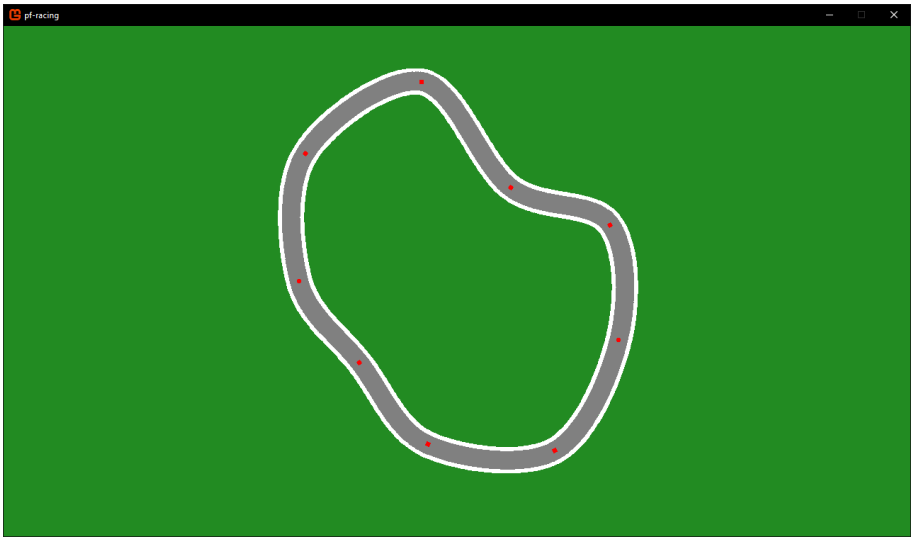

Figura 20 – Pista de corrida gerada com fator de estabilidade 1 e 9 pontos

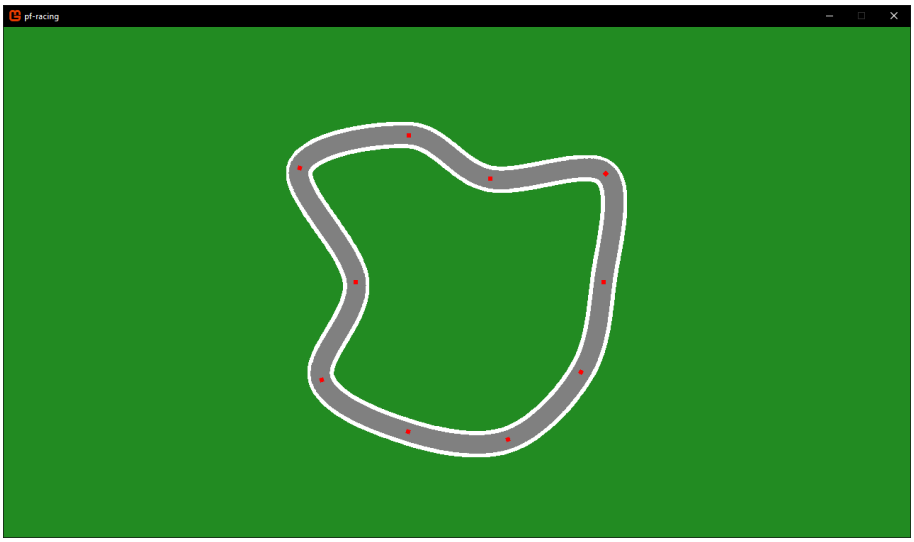

Figura 21 – Pista de corrida gerada com fator de estabilidade 0.9 e 10 pontos

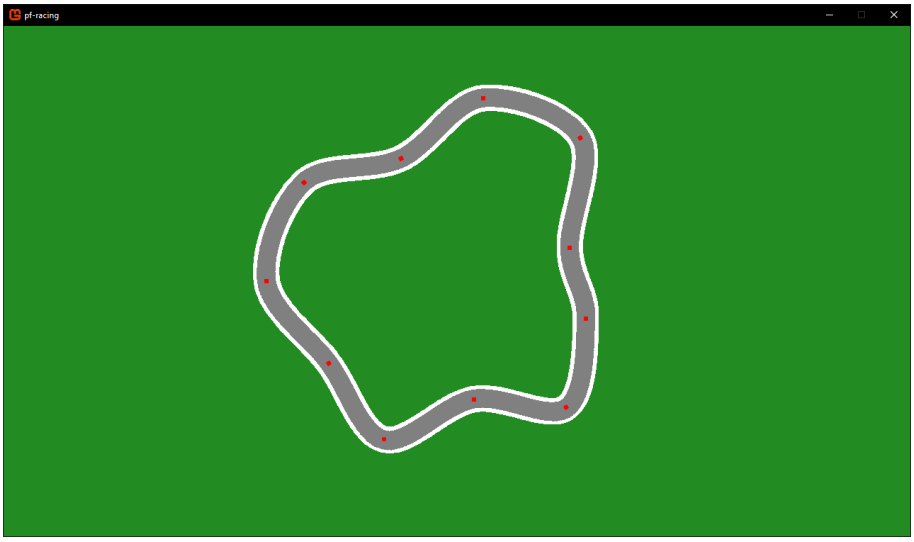

Figura 22 – Pista de corrida gerada com fator de estabilidade 1.2 e 11 pontos

5 Conclusão

O algoritmo apresentado nesse projeto pode ser utilizado para gerar proceduralmente pistas para um jogo digital, aumentando a diversidade e, consequentemente, trazendo uma maior sensação de divertimento por parte do jogador. Entretanto, ainda existe espaço para melhorias, principalmente acerca da etapa de validação da pista, evoluindo para um algoritmo *search-based*, onde os critérios são mais bem definidos e os resultados mais consistentes.

A geração de pista pode ser aperfeiçoada com o levantamento de estatísticas, como perfil de velocidade e curvatura. O perfil de velocidade diz respeito a velocidade média do carro em diferentes trechos da pista e o perfil de curvatura é o número de curvas presentes na pista, ambos serviriam como critérios para serem avaliados durante a parte de validação da pista. Desta maneira, gerar uma pista com um grau de dificuldade e diversidade especificado previamente seria possível e a consistência entre as pistas geradas seria maior do que o resultado apresentado.

Outro ponto para melhoria é a inteligência artificial, devido a maneira que utiliza para navegar, indo sempre em direção ao próximo *waypoint*, seu comportamento é mais distante do humano e não permite o desvio de obstáculos. Sendo assim, a validação feita por ela é muito dependente dos *waypoints* posicionados e o mal posicionamento dos mesmos pode comprometer a validação.

Em suma, o projeto mostrou uma maneira de gerar pistas de corrida, desde a representação das mesmas com o uso das curvas *Catmull-Rom* até o método radial e o uso de inteligência artificial para a geração procedural. Todavia, espaços para melhorias ainda existem para tornar ainda melhor o procedimento atual do projeto, evoluindo para uma geração de pistas mais personalizável e consistente.

Referências

1 NEWZOO. *Newzoo Global Games Market Report 2021 | Free Version*. Acesso em 13/11/2022. Disponível em: <https://newzoo.com/insights/trend-reports/newzoo-global-games-market-report-2021-free-version>. Citado na página 4.

2 SHAKER, N.; TOGELIUS, J.; NELSON, M. J. *Procedural Content Generation in Games*. [S.l.]: Springer, 2016. Citado na página 4.

3 CARDAMONE, L.; LOIACONO, D.; LANZI, P. L. Interactive evolution for the procedural generation of tracks in a high-end racing game. Association for Computing Machinery, New York, NY, USA, 2011. Disponível em: <https://doi.org/10.1145/2001576.2001631>. Citado 2 vezes nas páginas 4 e 5.

4 GAILLORETO, C. *History of Roguelike, from Rogue to Hades*. Acesso em: 13/11/2022. Disponível em: <https://screenrant.com/roguelike-definition-games-rogue-hades-roguelite-dungeon-crawler/>. Citado na página 4.

5 Supergiant Games. *Hades*. Acesso em 16/11/2022. Disponível em: <https://www.supergiantgames.com/games/hades/>. Citado na página 4.

6 The Game Awards. *2020 | Rewind | The Game Awards*. Acesso em 16/11/2022. Disponível em: <https://thegameawards.com/rewind/year-2020>. Citado na página 4.

7 MICROSOFT. *Site Oficial | Minecraft*. Acesso em 13/11/2022. Disponível em: <https://www.minecraft.net>. Citado na página 4.

8 ESRI. *Gerador de Cidade 3D processual: Projeto de Cidade em 3D para Ambientes urbanos*. Disponível em: <https://www.esri.com/pt-br/arcgis/products/arcgis-cityengine/overview>. Citado na página 5.

9 PIXAR. *Pixar's RenderMan | RenderMan Home*. Disponível em: <https://renderman.pixar.com/>. Citado na página 5.

10 RUSSELL, S.; NORVIG, P. *Artificial Intelligence: A Modern Approach*. [S.l.]: Pearson, 2016. Citado na página 5.

11 MONOGAME. Acessado: 30/06/2022. Disponível em: <https://www.monogame.net/>. Citado 2 vezes nas páginas 6 e 15.

12 MICROSOFT. *What is .NET Framework? A software development framework*. Acesso em: 30/06/2022. Disponível em: <https://dotnet.microsoft.com/en-us/learn/dotnet/what-is-dotnet-framework>. Citado na página 6.

13 TOGELIUS, J. et al. Search-based procedural content generation: A taxonomy and survey. *IEEE Transactions on Computational Intelligence and AI in Games*, v. 3, n. 3, p. 172–186, 2011. Citado na página 7.

14 TOGELIUS, J.; NARDI, R. D.; LUCAS, S. M. Towards automatic personalised content creation for racing games. *2007 IEEE Symposium on Computational Intelligence and Games*, 2007. Citado 2 vezes nas páginas 7 e 8.

15 PRASETYA, H. A.; MAULIDEVI, N. U. Search-based procedural content generation for race tracks in video games. *International Journal on Electrical Engineering & Informatics*, v. 8, n. 4, 2016. Citado 2 vezes nas páginas 7 e 8.

16 WANG, J.-Y.; LIN, Y.-B. Game ai: Simulating car racing game by applying pathfinding algorithms. *International Journal of Machine Learning and Computing*, IACSIT Press, v. 2, n. 1, p. 13, 2012. Citado na página 8.

17 DEROSE, T. D.; BARSKY, B. A. Geometric continuity, shape parameters, and geometric constructions for catmull-rom splines. Association for Computing Machinery, New York, NY, USA, v. 7, n. 1, p. 141, jan 1988. ISSN 0730-0301. Disponível em: <https://doi.org/10.1145/42188.42265>. Citado na página 12.

18 BREEN, W. R. D.; PEYSAKHOV, M. *Hermite and Catmull-Rom Curves*. Disponível em: <https://www.cs.drexel.edu/~david/Classes/CS536/Lectures/L-04_Hermite_Catmull_Rom.pdf>. Citado na página 12.

19 FARIN, G. Triangular bernstein-bézier patches. *Computer Aided Geometric Design*, Elsevier, v. 3, n. 2, p. 83–127, 1986. Citado na página 12.

20 LI, D.; HUANGFU, W.; LONG, K. Spline approximation-based data compression for sensor arrays in the wireless hydrologic monitoring system. *International Journal of Distributed Sensor Networks*, v. 13, p. 155014771769258, 02 2017. Citado na página 12.

21 MONOGAME Documentation. Acesso em: 30/06/2022. Disponível em: <https://docs.monogame.net/>. Citado na página 18.

22 MICROSOFT. *Visual Studio: IDE e Editor de Código para Desenvolvedores de Software e Teams*. Acesso em: 30/06/2022. Disponível em: <https://visualstudio.microsoft.com/pt-br/>. Citado na página 18.

23 HEILMEIER, A. *Racetrack database*. Acesso em: 08/09/2022. Disponível em: <https://github.com/TUMFTM/racetrack-database>. Citado na página 19.