Universidade de São Paulo
Instituto de Matemática e Estatística
Bacharelado em Ciência da Computação

RollerCoasterGenerator

desenvolvimento de um software de construção, simulação e geração de montanhas-russas virtuais voltado para jogos digitais

César Gasparini Fernandes

Monografia Final

mac 499— Trabalho de Formatura Supervisionado

Supervisor: Prof. Dr. Marcel P. Jackowski

São Paulo
5 de Fevereiro de 2022

Agradecimentos

Um espírito nobre engrandece o menor dos homens.
— Jebediah Obadiah Zachariah Springfield

A Deus, Jesus Cristo e Espírito Santo por tudo. Ao Prof. Dr. Marcel P. Jackowski por ser um excelente professor e por ter me orientado. À Priscila Gasparini Fernandes, ao Nelson de Jesus Fernandes, à Laura Helena Gasparini Fernandes, à Rosina Penha Lázzaro e ao Alexandre Gasparini Netto por todos os tipos de apoio que me dão. À Maria, ao Benedito, ao Joaguim, ao Joaguim, à Benedita, ao Sebastião, ao Manuel, à Laurinda e ao Afonso e a todos os outros por me guiarem. Aos professores do Instituto de Matemática e Estatística da Universidade de São Paulo por me ensinarem. Aos meus amigos pelo alívio dado durante a graduação.

Resumo

César Gasparini Fernandes. **RollerCoasterGenerator:** ***desenvolvimento de um software de construção, simulação e geração de montanhas-russas virtuais voltado para jogos digitais***. Monografia (Bacharelado). Instituto de Matemática e Estatística, Universidade de São Paulo, São Paulo, 2022.

As montanhas-russas são as principais atrações dos parques de diversão. Jogos de gerenciamento e simulação de parques de diversão, como RollerCoaster Tycoon® 2 e Planet Coaster®, permitem que o jogador construa e simule sua própria montanha-russa virtual. Entretanto, esses jogos não apresentam a possibilidade de gerar automaticamente uma montanha-russa dados alguns parâmetros de controle; apenas disponibilizam montanhas-russas pré-prontas para o jogador montar o seu parque. Tal ferramenta melhoraria a experiência do jogador e encurtaria o tempo entre a construção e a simulação, aumentando o divertimento. Portanto, o objetivo deste trabalho foi desenvolver um software que permite ao usuário construir ou gerar proceduralmente sua própria montanha-russa virtual além de simular um carro que trafegará ao longo do trajeto dadas as forças físicas atuantes. Adicionalmente, o software também permite a edição do relevo do solo e demais decorações do parque, como árvores e rochas. Para desenvolver tal software, foram modelados cortes transversais de trilhos que então foram extrusados ao longo de curvas de Bézier cúbicas parametrizadas de tal maneira que o usuário final possa ajustar a angulação do segmento de trilho. A simulação dos carros foi calculada utilizando as forças da gravidade, de atrito, de alavanca e de freio aplicadas ou não de acordo com o tipo de trilho que o carro está localizado. A velocidade foi aproximada utilizando Método de Runge-Kutta de quarto grau. Foram modelados e parametrizados trechos comumente presentes em montanhas-russas, como looping, queda e corkscrew, para serem utilizadas pelo gerador ao longo do trajeto base determinado pelo programador. O trajeto é formado por um conjunto de percursos que apresentam o tipo de trecho e suas restrições, como “trecho reto de tamanho entre 10 a 20 metros”. Cada tipo de percurso tem uma biblioteca dos trechos modelados que podem ser utilizados e cada trecho verifica a velocidade mínima que o carro precisa ter para completá-lo. Este projeto surgiu do desafio de desenvolver um gerador e simulador de montanhas-russas virtuais voltados a jogos virtuais e, por fim, foi desenvolvido tal software que atingiu o objetivo proposto e que satisfez os usuários que o testaram.

Palavras-chave: montanha-russa. gerador. extrusão. computação gráfica. procedural. bézier.

Abstract

César Gasparini Fernandes. **Proposta de Trabalho de Conclusão de Curso do César Gasparini Fernandes**. Capstone Project Report (Bachelor). Institute of Mathematics and Statistics, University of São Paulo, São Paulo, 2022.

Roller coasters are the main attractions of amusement parks. Amusement Park simulation and management games such as RollerCoaster Tycoon® 2 and Planet Coaster® allow the player to build and simulate their own virtual roller coaster. However, these games do not feature the ability to automatically generate a roller coaster given some control parameters; they only provide pre-made roller coasters for the player to set up their park. Such a tool would improve the player experience and shorten the time between build and simulation, increasing fun. Therefore, the objective of this work was to develop a software that allows the user to build or procedurally generate their own virtual roller coaster, in addition to simulating a car that will travel along the path given the acting physical forces. Additionally, the software also allows editing of the ground relief and other park decorations, such as trees and rocks. To develop such software, cross-sections of rails were modeled and then extruded along parameterized cubic Bézier curves in such a way that the end user can adjust the angulation of the rail segment. The car simulation was calculated using gravity, friction, lever and brake forces applied or not according to the type of track the car is located on. Velocity was approximated using the fourth degree Runge-Kutta method. Excerpts commonly present in roller coasters, such as looping, fall and corkscrew, were modeled and parameterized to be used by the generator along the base path determined by the programmer. The route is formed by a set of routes that present the type of section and its restrictions, such as "straight section with a size between 10 and 20 meters". Each route type has a library of modeled sections that can be used and each section checks the minimum speed the car needs to complete it. This project arose from the challenge of developing a virtual roller coaster generator and simulator aimed at virtual games and, finally, such software was developed that reached the proposed objective and satisfied the users who tested it.

Keywords: roller coaster. generator. extrusion. computer graphics. procedural. bézier.

Lista de Figuras

Lista de Algoritmos

Sumário

Apêndices

Capítulo 1

Introdução

1.1 Motivação

Inspirada nos escorregadores de gelos populares na Rússia nos séculos 16 e 17, a primeira montanha-russa moderna do mundo foi inaugurada em 1817 na França, como dito em Harris e Threewitt, 2007. Desde então, as montanhas-russas foram evoluindo e se tornando cada vez mais populares, sendo as atrações principais dos parques de diversão a partir dos anos 80 e chamando a atenção dos jovens.

Graças à alta popularidade das montanhas-russas, algumas empresas de jogos eletrônicos, cujo público alvo é principalmente adolescente, começaram a inseri-las em seus jogos para conquistar uma maior parcela do mercado. A empresa Sega Corporation, para competir com a Nintendo Co., Ltd. na década de 90, criou o jogo Sonic the Hedgehog™, o qual tem a jogabilidade e os mapas baseados nas montanhas-russas japonesas, como mostrado em Acks *et al.*, 2020.

Jogos de gerenciamento e simulação de parques de diversão, como RollerCoaster Tycoon® 2 e Planet Coaster®, permitem que o jogador construa e simule sua própria montanha-russa virtual. Tais jogos disponibilizam uma série de objetos e ferramentas para o usuário decorar o terreno ao redor de suas montanhas-russas para que ele possa personalizá-la ao máximo possível.

Entretanto, nenhum jogo famoso de construção e simulação de parques de diversão apresenta a possibilidade de gerar automaticamente uma montanha-russa dados alguns parâmetros de controle; apenas disponibilizam montanhas-russas pré-prontas para o jogador montar o seu parque. Tal ferramenta melhoraria a experiência do jogador e encurtaria o tempo entre a construção e a simulação, aumentando o divertimento.

Portanto, este projeto surge do desafio de desenvolver um gerador e simulador de montanhas-russas virtuais voltados a jogos digitais. Assim como os construtores e simuladores de montanhas-russas dos principais jogos de parques de diversão, este trabalho não tem o escopo de modelar ou simular montanhas-russas reais. No entanto, a geração automática de trechos ao longo do trajeto, respeitando os limites físicos, e grau de diversão prescritos, podem até sugerir futuros modelos de montanhas-russas reais.

1.2 Objetivo

Desenvolver uma aplicação que permita ao usuário construir ou gerar proceduralmente sua própria montanha-russa virtual além de simular um carro que trafegará ao longo do trajeto dadas as forças físicas atuantes. Adicionalmente, o software também permitirá a edição do relevo do solo e demais decorações do parque, como árvores e rochas.

1.3 Organização do texto

No capítulo 2, é explicado como foram feitos o construtor, simulador e gerador de montanhas-russas, o terreno e os objetos de decoração e a interface gráfica do programa. No capítulo 3, são apresentados os resultados e, no capítulo 4, é concluído o trabalho. Por fim, no apêndice A, está o manual.

Capítulo 2

Metodologia

O desenvolvimento do trabalho pode ser dividido em cinco etapas: construtor; simulador; gerador; terreno e interface. Cada uma delas será explicada sucintamente e, em seguida, aprofundada nas próximas seções. O texto desse capítulo foi escrito da forma *bottom-up*, ou seja, o raciocínio é construído do detalhe para o todo, partindo dos vértices da modelagem dos trilhos até a geração da montanha-russa, subindo os níveis de abstração.

O programa foi desenvolvido utilizando a engine Unity® por apresentar o pipeline de renderização pronto e por ser comumente empregado em desenvolvimento de jogos no mercado. Porém, para o desenvolvimento das partes principais do projeto, que são o construtor, o simulador e o gerador, foi utilizado apenas o básico de suas bibliotecas, como os objetos "Vector3", "Matrix4x4", "Mesh", entre outros. Já para a interface gráfica, a biblioteca de interface gráfica do Unity® foi utilizada massivamente, dado que, para ajustar as posições, animações e eventos dos elementos da interface (como botões e painéis), é quase imprescindível utilizar a biblioteca.

Para o desenvolvimento do construtor de montanhas-russas, foi utilizada uma técnica semelhante à de Hyková (2010) para o trajeto dos trilhos. Foram modelados cortes transversais de tipos de trilhos de montanhas-russas que são extrusados ao longo de Curvas de Bézier cúbicas definidas pelo jogador através da interface gráfica, o qual pode ajustar a elevação, a rotação, a inclinação e o comprimento do segmento de trilho que está construindo de maneira semelhante ao ajuste dos parâmetros no construtor presente no Planet Coaster®. Estas curvas são "ligadas" de maneira que seus pontos de controle tornem o trajeto inteiro contínuo. Também foram desenvolvidas algumas ferramentas como a de autocompletar a montanha-russa e a de checar se os trilhos estão se intersectando. Por fim, foram disponibilizados trechos pré-prontos de trilhos, como montanha ou looping, ao usuário para facilitar a construção.

Para o desenvolvimento do simulador dos vagões, são calculadas as diferentes forças aplicadas ao vagão de acordo com o tipo de trilho que ele está. Por exemplo, em um trilho do tipo "normal", são aplicadas a força da gravidade e do atrito ao vagão, já em um trilho do tipo "alavanca", também é aplicada a força da alavanca. A simulação do carro no trilho é realizada utilizando o Método de Runge-Kutta de quarto grau para aproximar sua velocidade entre dois momentos curtos distintos (quadros do programa). Também são

pré-computados mapas de calor da velocidade, da força-g e da altura dos vagões ao longo do trilho a fim de auxiliar o jogador na construção de sua montanha-russa.

O gerador de montanhas-russas gera os trilhos proceduralmente, se baseando em um L-System estocástico paramétrico, como o de Hanan, 1992, respeitando as seguintes restrições:

- Os trilhos não podem se intersectar;
- O percurso deve ser circular, ou seja, a montanha-russa deve "fechar";
- O carro deve completar o percurso.

Inicialmente o gerador recebe um trajeto, criado pelo programador, divididos em percursos de diferentes tipos e restrições, como "reto de tamanho entre 20 a 30 metros" e "curva de 90° à direita". Para cada segmento, ele verifica a velocidade que o carro teria, escolhe um trecho pré-pronto de trilho do tipo de percurso atual e que aceite a velocidade do carro, ajusta seus parâmetros para que o carro consiga completá-lo e o gera. No final, é utilizada a ferramenta de autocompletar para "fechar" o circuito da montanha-russa gerada.

Para o desenvolvimento da ferramenta de edição de terreno, foi gerada uma malha triangular para o terreno passível de deformações pelo usuário para poder criar aclives e declives. Também foi desenvolvida uma ferramenta que possibilita ao jogador colocar alguns objetos, como paredes e torres de castelo, ao longo do terreno. Esses objetos são modelos de domínio público.

A interface gráfica foi prototipada utilizando o programa diagrams.net® e, em seguida, foi desenvolvida utilizando a biblioteca de interface gráfica do Unity®. Na interface, estão disponíveis informações estruturais do segmento de trilho atual, como elevação, rotação, inclinação e tamanho, informações físicas dos vagões no trilho, como velocidade e força-g, elementos que auxiliam o usuário na construção, como setas para rotacionar o trilho e botões que permitem que o jogador possa acessar os recursos do programa, como salvar, carregar, construir e gerar uma montanha-russa.

2.1 Construtor

Para o jogador construir sua montanha-russa, foi definido que ele pode colocar segmentos de trilho um seguido do outro, podendo ajustar os ângulos de elevação, de rotação e de inclinação, o comprimento e o tipo de cada segmento. Os trilhos podem ser de 4 tipos: plataforma; normal; alavanca e freios. Também foi definido que um segmento de trilho deve ter uma orientação inicial e seu percurso deve ser um arco ou um segmento de reta para facilitar a construção e a manipulação dos parâmetros.

A geração dos modelos tridimensionais dos trilhos foi feita por meio da extrusão poligonal, que consiste, basicamente, de transformar um conjunto de pontos em um plano bidimensional ao longo de uma curva tridimensional. Os conjuntos de pontos a serem extrusados são os modelos feitos dos cortes transversais dos trilhos e as curvas da extrusão são os percursos dos segmentos de trilhos. Portanto, será apresentado como foram executadas as modelagens dos cortes transversais, em seguida, será explicado como foram

calculadas as funções matemáticas dos percursos dos segmentos de trilho e, finalmente, será aprofundada a realização da extrusão.

2.1.1 Modelagem dos cortes transversais

Foram modelados dois trilhos diferentes nesse projeto, um baseado no trilho *Vekoma new spine* e outro baseado no trilho *Intamin 3-pipe*, ambos presentes em Väisänen (2018) e ilustrados na figura 2.1. Os modelos foram divididos em partes mais simples para que a triangulação dos pontos extrusados seja mais simples. Por exemplo, o modelo *Vekoma new spine* foi dividido em três partes: bases dos cilindros, lados dos cilindros e *sleepers* (partes transversais). As separações estão ilustradas na figura 2.2. Também foram modeladas partes adicionais aos trilhos para representarem os quatro tipos diferentes de trilho, como mostrado na figura 2.3.

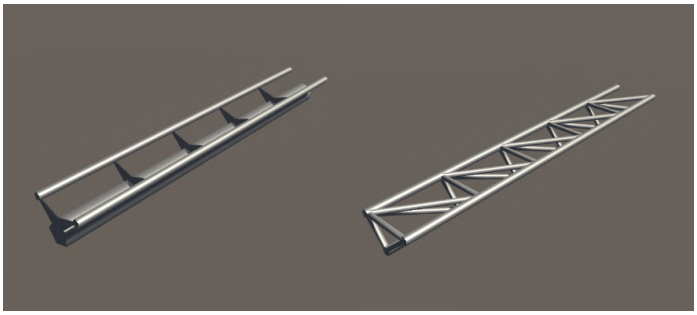


Figura 2.1: *Modelos de trilho feitos nesse trabalho. À esquerda, está o modelo baseado no trilho Vekoma new spine e, à direita, está o modelo baseado no trilho Intamin 3-pipe.*

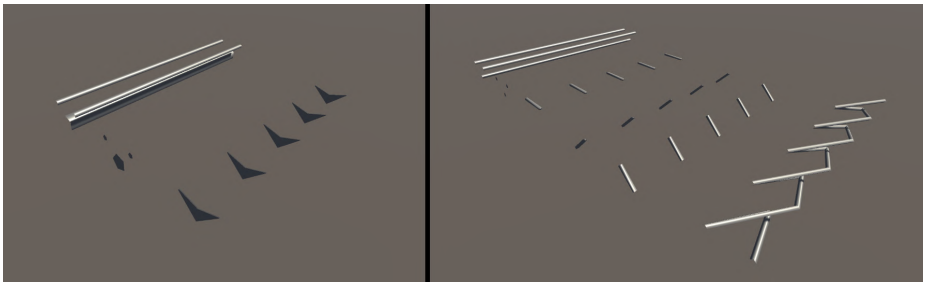


Figura 2.2: *Separação dos modelos de trilho em partes mais simples. À esquerda, estão as partes do modelo baseado no trilho Vekoma new spine e, à direita, estão as partes do modelo baseado no trilho Intamin 3-pipe. As faces que não estão direcionadas à câmera não estão visíveis por causa do pipeline de renderização.*

Para serem modelados os cortes transversais dos trilhos, é necessário definir os eixos que os tornam transversais. Seja um carro trafegando ao longo do trilho e seja β a matriz que representa sua base em um instante. Logo, o vetor que representa a direção do carro, casso ele fosse para frente, $\beta.x$, é chamado de vetor tangente e a intersecção do plano definido por ele e o trilho forma o corte transversal do trilho. O vetor que parte do chão do carro e vai até o encosto da cadeira, representando a vertical local do carro, $\beta.y$, é chamado de vetor vertical. Por fim, $\beta.z$ é chamado de vetor binormal. A base está ilustrada na figura 2.4.

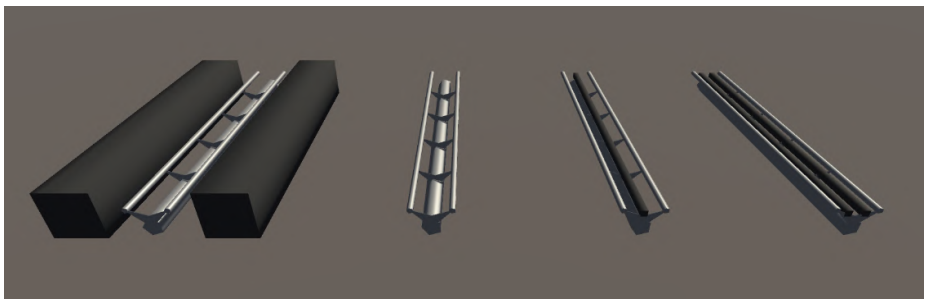


Figura 2.3: *Modelos de diferentes tipos de trilho do modelo Vekoma new spine. Da esquerda para a direita estão os tipos: plataforma; normal; alavanca; freios.*

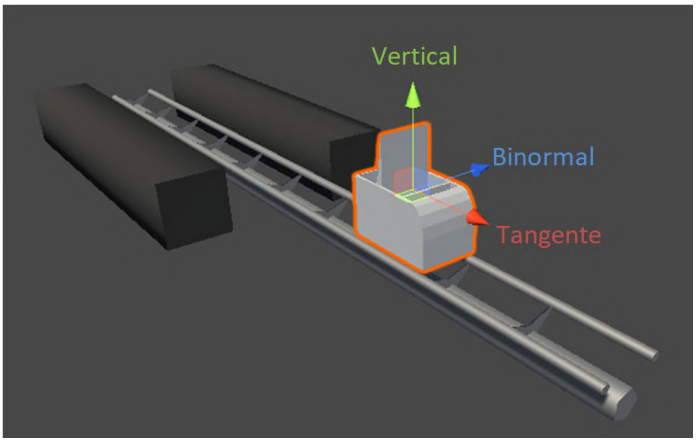


Figura 2.4: *Vetores tangente, vertical e binormal do carro em um instante ao longo do trilho.*

Com o plano do corte transversal definido, foram modelados os vértices dos polígonos formados. Como há algumas estruturas do trilho que não são contínuas ao longo do trajeto, como os *sleepers*, foram modelados apenas uma cópia da parte que se repete para replicá-la ao longo do trilho. A fim de aumentar a eficiência do programa, os círculos foram aproximados por pentágonos pois apresentam menos faces para o pipeline de renderização processar. Um exemplo de modelagem dos vértices está presente na figura 2.5. Para o cálculo do sombreamento dos modelos pelo pipeline de renderização, também foi necessário modelar as normais dos vértices, estando exemplificadas na figura 2.6.

2.1.2 Cálculo das trajetórias dos segmentos de trilho

Para modelar as trajetórias dos segmentos de trilho em funções, é conveniente fazer o uso de Splines por ter um fácil controle de funções no espaço. Mais especificamente, será feito o uso de curvas de Bézier cúbicas por serem comumente utilizadas em trabalhos de computação gráfica. Hyková (2010) também fez uso de curvas de Bézier cúbicas para modelar as trajetórias dos trilhos de suas montanhas-russas, porém sua parametrização foi diferente das adotadas neste trabalho.

Como definida em Lengyel (2011), uma curva de Bézier cúbica é um polinômio de terceiro grau determinada por quatro pontos chamados de pontos de controle. Sejam $\mathbf{P}_0$, $\mathbf{P}_1$, $\mathbf{P}_2$, $\mathbf{P}_3$ os pontos de controle nessa ordem, a curva de Bézier, $\mathbf{B}(t)$, é definida como a equação 2.1.

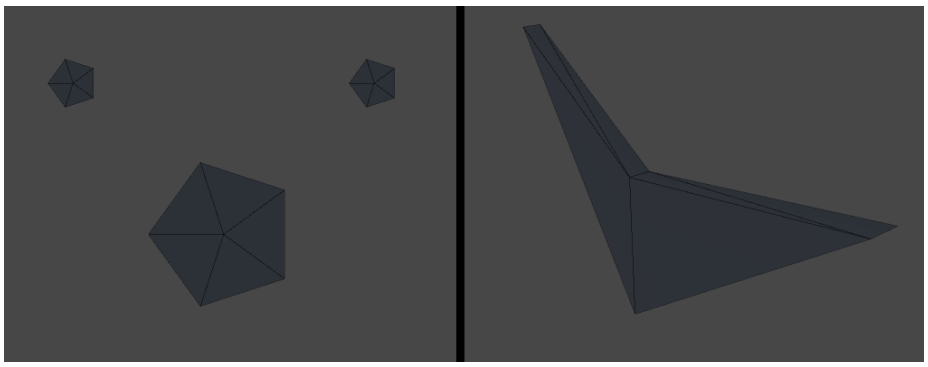


Figura 2.5: *Representação dos vértice dos cortes transversais das partes do modelo do trilho Vekoma new spine. À esquerda estão os vértices do corte das bases e dos lados dos cilindros e à direita está o modelo de um sleeper. A triangulação dos vértice do modelo foram feitas para facilitar a visualização da figura.*

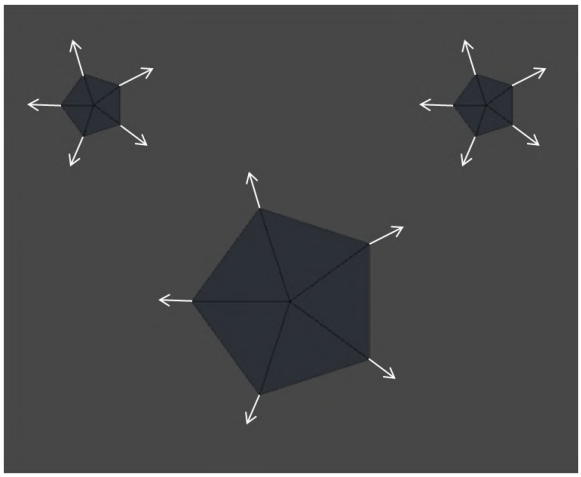


Figura 2.6: *Representação das normais em uma das partes do modelo do trilho Vekoma new spine. As normais estão representadas em branco e estão fora de escala. A triangulação dos pontos do modelo foram feitas para facilitar a visualização da figura.*

$$\mathbf{B}(t) = (1-t)^3\,\mathbf{P}_0 + 3t\,(1-t)^2\,\mathbf{P}_1 + 3t^2\,(1-t)\,\mathbf{P}_2 + t^3\mathbf{P}_3 \tag{2.1}$$

Dado que $\mathbf{B}(t)$, $t \in [0,1]$, descreve a trajetória do segmento de trilho, têm-se que $\mathbf{B}(0)$ e $\mathbf{B}(1)$ são os pontos inicial e final do segmento, respectivamente, e, também, $\mathbf{P}_0$ e $\mathbf{P}_3$ são os pontos inicial e final do segmento, respectivamente. A tangente de $\mathbf{B}(t)$ pode ser escrita como a equação 2.2.

$$\mathbf{B}'(t) = 3(1-t)^2(\mathbf{P}_1 - \mathbf{P}_0) + 6(1-t)t(\mathbf{P}_2 - \mathbf{P}_1) + 3t^2(\mathbf{P}_3 - \mathbf{P}_2) \tag{2.2}$$

Para a curva de Bézier se adequar a restrição de que os segmentos de trilho devem ser arcos, elas devem ter seus pontos de suporte dispostos de tal maneira que a curva se aproxime de um arco. Como $\mathbf{P}_0$ é o ponto inicial do segmento de trilho, ele já está definido. Suponha que o ponto e $\mathbf{P}_3$ já estejam definidos por ser o final do arco, então,

apenas os pontos $\mathbf{P}_1$ e $\mathbf{P}_2$ devem ser calculados. A aproximação feita por Rišкus (2006) foi adaptada para ser utilizada nesse trabalho. Seja r o raio do círculo que contém $\mathbf{P}_0$ e $\mathbf{P}_3$ e esteja centrado na intersecção das retas $\mathbf{P}_0 + t\mathbf{B}'(0) = 0, t \in \mathbb{R}$, e $\mathbf{P}_3 + t\mathbf{B}'(1) = 0, t \in \mathbb{R}$, como representado na figura 2.7. Logo, os pontos $\mathbf{P}_1$ e $\mathbf{P}_2$ são definidos nas equações 2.3 e 2.4, respectivamente. Note que, mesmo $\mathbf{B}(t)$ não estando definida ainda, é sabido $\mathbf{B}'(0)$ e $\mathbf{B}'(1)$ pois $\mathbf{B}(t)$ é um arco e as tangentes são referentes a pontos conhecidos.

$$\mathbf{P}_1 = \mathbf{P}_0 + 0.55191496 \cdot r \cdot \mathbf{B}'(0) \tag{2.3}$$

$$\mathbf{P}_2 = \mathbf{P}_3 - 0.55191496 \cdot r \cdot \mathbf{B}'(1) \tag{2.4}$$

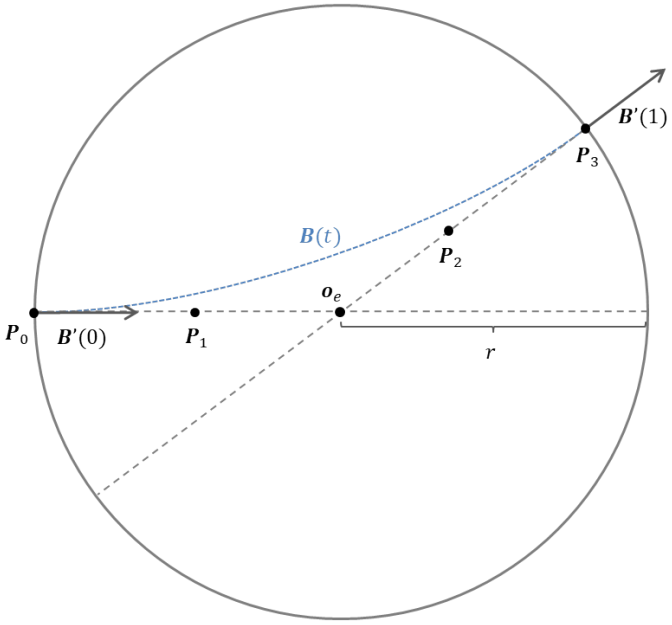


Figura 2.7: *Aproximação de um arco por uma Bézier.* $\mathbf{B}(t), t \in [0, 1]$ *é a Bézier,* $\mathbf{P}_0$, $\mathbf{P}_1$, $\mathbf{P}_2$, $\mathbf{P}_3$ *são seus pontos de suporte,* $\mathbf{o}_e$ *é a intersecção das retas* $\mathbf{P}_0 + t\mathbf{B}'(0) = 0, t \in \mathbb{R}$, *e* $\mathbf{P}_3 + t\mathbf{B}'(1) = 0, t \in \mathbb{R}$ *e o centro do círculo formado pelos pontos* $\mathbf{P}_0$ *e* $\mathbf{P}_3$ *com raio* r.

Embora esteja definida a função das trajetórias dos segmentos de trilho, é necessário parametrizá-la para se obter $\mathbf{P}_3$ e r. O que se quer é uma função que, dado o ponto inicial, a orientação inicial, o ângulo de elevação, o ângulo de rotação e o comprimento do segmento, ela retorne uma curva de Bézier cúbica que descreva o trajeto desejado.

2.1.3 Parametrização da trajetória dos segmentos de trilho

A fim de se obter um segmento de trilho com o percurso de um arco, foi construída uma esfera, chamada de Esfera Imaginária, para auxiliar na parametrização da trajetória. O motivo da esfera ter esse nome é por não existir no ambiente e por ser apenas um conceito que dá suporte nos cálculos.

Seja $\mathbf{B}(t)$ a função de percurso do segmento de trilho que tem elevação e rotação iguais a p_e e p_r, respectivamente.Seja $\mathbf{t}_0$ e $\mathbf{t}_1$ as tangentes nos pontos iniciais e finais do percurso

do segmento de trilho e $\mathbf{u}_0$ e $\mathbf{u}_1$ os vetores verticais nos pontos iniciais e finais do percurso. Seja r e $\mathbf{o}_e$ o raio e o centro da Esfera Imaginária, respectivamente. Portanto, a Esfera Imaginária tem as seguintes propriedades:

- Todos os pontos do percurso do segmento de trilho estão contidos na esfera;
- Os pontos inicial e final do segmento de trilho estão contidos em sua casca esférica;
- A tangente do ponto inicial do percurso é ortogonal à sua casca esférica no ponto inicial;
- A tangente do ponto final do percurso é ortogonal à sua casca esférica no ponto final;
- $\mathbf{B}(0) + r \cdot \mathbf{t}_0 = \mathbf{o}_e$;
- $\mathbf{B}(1) - r \cdot \mathbf{t}_1 = \mathbf{o}_e$.

Portanto temos uma esfera igual a mostrada na figura 2.8.

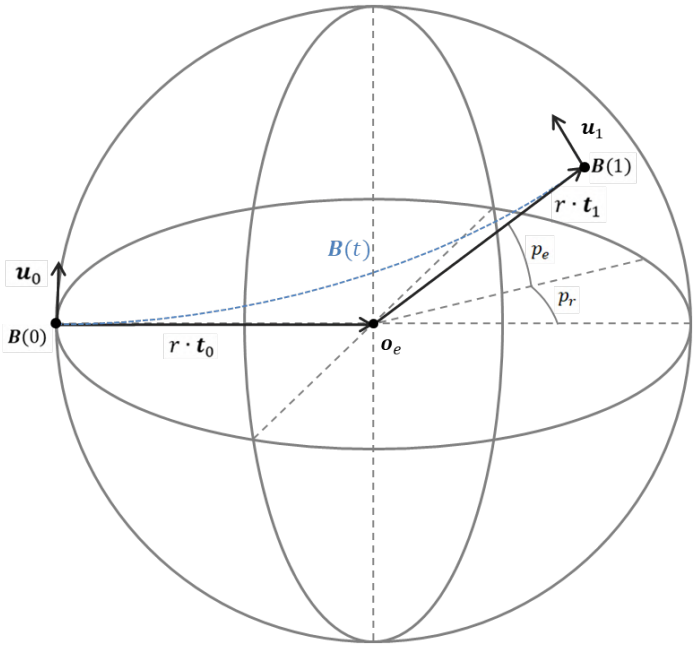


Figura 2.8: *Esquema de uma Esfera Imaginária.* $\mathbf{B}(t), t \in [0, 1]$ *é a Bézier que descreve o trajeto do segmento de trilho,* $\mathbf{o}_e$ *é o centro da Esfera Imaginária,* p_r *é o ângulo de rotação do segmento de trilho,* p_e *é o ângulo de elevação do segmento de trilho,* $\mathbf{u}_0$ *é o vetor vertical da base inicial da Bézier,* $\mathbf{t}_0$ *é o vetor tagente da base inicial da Bézier,* $\mathbf{u}_1$ *é o vetor vertical da base final da Bézier,* $\mathbf{t}_1$ *é o vetor tangente da base final da Bézier.*

Com a Esfera Imaginária construída, é possível calcular o ponto $\mathbf{B}(1)$. Dada a base inicial, β_i, e o ponto $\mathbf{B}(0)$, $\mathbf{B}(1)$ é calculado seguindo o algoritmo 1. Nesse trabalho, para rotacionar um vetor ao redor de um eixo ou para calcular a matriz de rotação ao redor de um eixo, foi utilizada a biblioteca de quatérnios do Unity®.

Algoritmo 1: Cálculo de $\mathbf{B}(1)$

Entrada: a base inicial, β_i, e o ponto $\mathbf{B}(0)$
Saída: o ponto $\mathbf{B}(1)$

início
  Seja **deslocamento** $\longleftarrow r \cdot \mathbf{t}_0$, o vetor de suporte;
  Seja **proj** $\longleftarrow (\beta_i.x.x, 0, \beta_i.x.z)$, o vetor de projeção;
  proj $\longleftarrow \frac{\mathbf{proj}}{\|\mathbf{proj}\|}$;
  se $\beta_i.y.y < 0$ **então**
    proj $\longleftarrow -\mathbf{proj}$;
  fim
  Seja $\mathbf{eixo}_1 \longleftarrow \mathbf{proj} \times (0, 1, 0)$, o eixo de rotação;
  se $\|\mathbf{eixo}_1\| < 1$ **então**
    $\mathbf{eixo}_1 \longleftarrow \beta_i.z$;
  fim
  Rotacione **deslocamento** ao redor de $\mathbf{eixo}_1$, p_e radianos;
  Rotacione **deslocamento** ao redor de $(0, 1, 0)$, p_r radianos;
  Seja $\mathbf{B}(1) \longleftarrow r\mathbf{B}(0) + \cdot\mathbf{t}_0 + r \cdot$ **deslocamento**;
  Retorne $\mathbf{B}(1)$;
fim

O algoritmo 1, basicamente, rotaciona $\mathbf{t}_0$ ao redor de $\beta_i.z$ (desconsiderando a inclinação do segmento) em direção a $\beta_i.y$, o "elevando" em relação ao início do segmento, e, em seguida, o rotaciona ao redor do eixo y, o "rotacionando" em relação ao início do segmento. Por fim, esse vetor rotacionado é multiplicado por r e somado com $\mathbf{B}(0) + r \cdot \mathbf{t}_0$ para se obter o ponto final.

Tendo $\mathbf{B}(1)$ calculado, é necessário realizar o cálculo do raio da esfera. Seja $\mathbf{t}_0^{'}$ o vetor $\mathbf{t}_0$ após as transformações descritas anteriormente. Note que $\mathbf{t}_0^{'} = \mathbf{t}_1$.Seja α o ângulo entre $\mathbf{t}_0$ e $\mathbf{t}_0^{'}$, portanto $\alpha = acos\left(\mathbf{t}_0 \cdot \mathbf{t}_0^{'}\right)$. Seja l o comprimento da trajetória do segmento do trilho. Caso $\alpha = 0$, trivialmente, $r = \frac{l}{2}$, pois, o segmento de trilho será uma reta. Seja $0 < \alpha \leq \frac{\pi}{2}$ e considere o corte planar da esfera contendo $\mathbf{B}(0)$, $\mathbf{B}(0) + r\mathbf{t}_0$ e $\mathbf{B}(1)$ como mostrado na figura 2.9.Note que o percurso do trilho também está contido nesse plano. Considere o círculo de raio r que está contido no plano e na esfera e note que $\mathbf{o}_e$ é o seu centro.

Como o percurso é um semi arco, considere o círculo que ele está contido. Seja R e $\mathbf{o}_c$ o raio e o centro desse círculo, respectivamente. Portanto, pela relação entre arco e círculo, têm-se a equação 2.5.

$$\frac{2\pi R}{l} = \frac{2\pi}{\alpha} \iff R = \frac{l}{\alpha} \tag{2.5}$$

Considerando o triângulo retângulo $\mathbf{B}(0)\mathbf{o}_e\mathbf{o}_c$, mostrado na figura 2.10, têm-se a equação 2.6.

$$tan\left(\frac{\alpha}{2}\right) = \frac{r}{R} \Longrightarrow r = R \cdot tan\left(\frac{\alpha}{2}\right) \iff r = \frac{l \cdot tan\left(\frac{\alpha}{2}\right)}{\alpha} \tag{2.6}$$

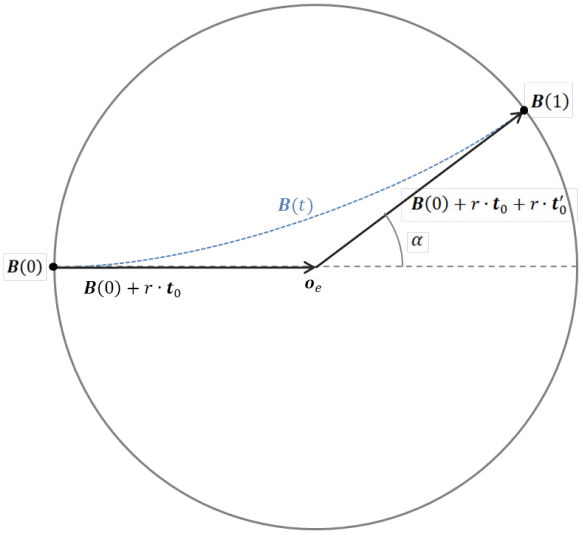


Figura 2.9: *Esquema do corte do círculo da Esfera Imaginária contendo* $\mathbf{B}(0)$, $\mathbf{B}(0) + r\mathbf{t}_0$ *e* $\mathbf{B}(1)$, *sendo que* α *é o ângulo entre* $\mathbf{t}_0$ *e* $\mathbf{t}_0^{'}$, $\mathbf{o}_e$ *é o centro do círculo e* r *é o raio do círculo.*

Portanto, o raio da esfera imaginária pode ser calculado segundo a equação 2.7.

$$r = \begin{cases} \frac{l}{2}, & \alpha = 0 \\ \frac{l \cdot tan\left(\frac{\alpha}{2}\right)}{\alpha}, & 0 < \alpha \leq \pi \end{cases} \tag{2.7}$$

Como estão definidos os cálculos da trajetória dos segmentos de trilho, é possível extrusar os cortes transversais modelados na subseção 2.1.1. Na próxima subseção, será aprofundada a extrusão.

2.1.4 Extrusão dos modelos dos cortes transversais

Semelhante a forma descrita informalmente por Park (2003), a extrusão poligonal recebe um polígono e uma trajetória sem intersecções consigo mesma e gera um conjunto de triângulos definindo o volume gerado pelas transformações do conjunto de pontos ao longo da trajetória, mantendo, sempre, o plano contendo o polígono ortogonal à tangente da trajetória. Neste trabalho, os modelos foram transformados a uma distância uniforme e os pontos gerados foram triangulados.

Como há diferentes modelos a serem extrusados, a triangulação dos pontos extrusados foram separados do algoritmo de extrusão pois cada modelo tem sua própria forma de triangulá-los. Portanto, a extrusão, nesse trabalho, apenas transforma o modelo recebido ao longo do percurso, retornando os novos pontos a serem triangulados, como no algoritmo simplificado 2. O algoritmo do código da extrusão desse trabalho também transforma as normais dos modelos, porém, para simplificar o texto, elas serão omitidas nos algoritmos por serem análogas aos pontos.

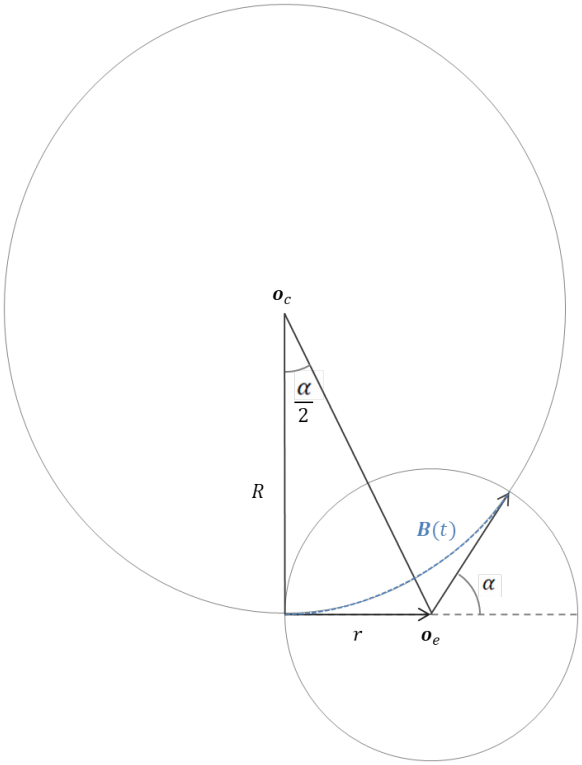


Figura 2.10: *Esquema para o cálculo do raio da Esfera Imaginária, dados o raio, R, da esfera que contém o arco aproximado pela função* $\mathbf{B}(t)$, $t \in [0,1]$ *e o ângulo entre os vetores tangentes da base inicial e a base final, sendo* $\mathbf{o}_e$ *o centro da Esfera Imaginária e* $\mathbf{o}_c$ *o centro do círculo de raio R.*

Algoritmo 2: Algoritmo de extrusão (simplificado)

Entrada: o vetor de pontos, $\mathbf{pontos}_{iniciais}$, a base inicial, β_i, o trajeto, $\mathbf{B}(t)$, $t \in [0,1]$, a elevação, p_e, a rotação, p_r, a inclinação, p_i, o comprimento, l, e a resolução, res

Saída: o vetor de vetores de pontos extrusados

início
- Seja $\mathbf{Transf}_i$ a matriz de mudança de base da matriz identidade para β_i;
- **para cada ponto** *em* $\mathbf{pontos}_{iniciais}$ **faça**
  - $\mathbf{ponto} \longleftarrow \mathbf{Transf}_i \cdot \mathbf{ponto}$;
- **fim**
- Seja $\mathbf{pontos}_{extrusados}[res + 1][\mathbf{pontos}_{iniciais}.tamanho]$;
- Calcule a base final, β_f;
- Seja $\Delta S \longleftarrow \frac{l}{res}$;
- Seja $t \longleftarrow 0$;
- **para** $i \longleftarrow 0$ **até** res **por** 1 **faça**
  - Seja **RotMatriz** a matriz de rotação da base β_i para a base interpolada entre β_i a β_f por t;
  - **para** $j \longleftarrow 0$ **até** $\mathbf{pontos}_{iniciais}[i].tamanho$ **por** 1 **faça**
    - $\mathbf{pontos}_{extrusados}[i][j] \longleftarrow (\mathbf{RotMatriz} \cdot \mathbf{pontos}_{iniciais}[j]) + \mathbf{B}(t)$;
  - **fim**
  - $t \longleftarrow t + \Delta t$, tal que $\Delta S = \int_t^{t+\Delta t} \mathbf{B}(x)\, dx$;
- **fim**
- retorne $\mathbf{pontos}_{extrusados}$;

fim

É necessário definir os cálculos de β_f, **RotMatriz** e Δt. O cálculo da base final, β_f, é feito estendendo o algoritmo 1. Sumariamente, a base inicial, β_i, é rotacionada ("elevada") p_e radianos ao redor de $\beta_i.z$ e "em direção" a $\beta_i.y$, em seguida, rotacionada p_r radianos ao redor do eixo y e, por fim, rotacionada ("inclinada") p_i radianos ao redor do novo $\beta'_i.x$. O cálculo da matriz de rotação, $Transf_\beta$, tal que $\beta_f = Transf_\beta \cdot \beta_i$, é feito seguindo o algoritmo 3.

Algoritmo 3: Cálculo da matriz de rotação, $Transf_\beta$

Entrada: a base inicial, β_i, a elevação, p_e, a rotação p_r, a inclinação p_i
Saída: matriz de rotação, $Transf_\beta$

```
1  início
2      Seja x, y, z, os vetores β_i.x, β_i.y, β_i.z, respectivamente;
3      Seja proj ⟵ (x.x, 0, x.z), o vetor de projeção;
4      proj ⟵ proj / ‖proj‖;
5      se y.y < 0 então
6          proj ⟵ −proj;
7      fim
8      Seja eixo_1 ⟵ proj × (0, 1, 0), o eixo de rotação;
9      se ‖eixo_1‖ < 1 então
10         eixo_1 ⟵ z;
11     fim
12     Seja M_e a matriz de rotação de p_e radianos ao redor de eixo_1;
13     x ⟵ M_e · x;
14     Seja M_r a matriz de rotação de p_r radianos ao redor de (0, 1, 0);
15     x ⟵ M_r · x;
16     Seja M_i a matriz de rotação de p_i radianos ao redor de x;
17     Retorne M_i · M_r · M_e;
18 fim
```

A fim de calcular a interpolação entre β_i e β_f, foram feitas algumas convenções para as matrizes resultantes respeitarem a função da trajetória. Seja $\beta_t, t \in [0, 1]$, a base no instante t da interpolação. Tem-se que $\beta_t.x = \mathbf{B}'(t)$, dado que $\mathbf{B}'(t)$ é a derivada da trajetória do segmento de trilho no instante t. Como se têm um dos vetores de β_t e por ela ser ortonormal, basta realizar o cálculo de outro de seus vetores.

Seja $R(\theta, \mathbf{v})$, $\theta \in \mathbb{R}$ e $\mathbf{v} \in \mathbb{R}^3$, a função que retorna a matriz de rotação de θ radianos ao redor do eixo $\mathbf{v}$. Seja $\beta'_i = R\left(\beta_i.x \times \beta_f.x, \beta_i.x \cdot \beta_f.x\right) \cdot \beta_i$, ou seja, β_i rotacionada para que $\beta'_i.x = \beta_f.x$. Seja φ o ângulo em radianos de $\beta'_i.y$ a $\beta_f.y$, logo, se β'_i for rotacionada φ radianos ao redor de $\beta_f.x$ em direção a $\beta_f.y$, ambas as bases serão iguais. Portanto, para interpolar as bases β_i e β_f no instante t, basta rotacionar β_i para que $\beta'_i.x = \mathbf{B}'(t)$ e, em seguida, rotacionar β'_i em $t \cdot \varphi$ radianos ao redor de $\beta'_i.x$. Um esquema dessas operações está representado na figura 2.11. O algoritmo para encontrar φ está descrito em 4 e o algoritmo para o cálculo da matriz de rotação da interpolação no instante t está em 5.

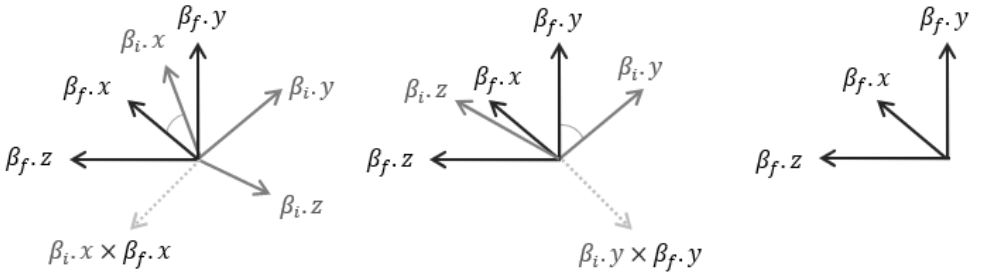


Figura 2.11: *Esquema das etapas de transformação da base β_i para a base β_f.*

Algoritmo 4: Cálculo de φ da interpolação entre β_i e β_f

Entrada: a base inicial, β_i, a base final, β_f
Saída: o ângulo φ em radianos

início
  Seja $\mathbf{x}_i$, $\mathbf{y}_i$, os vetores $\beta_i.x$, $\beta_i.y$, respectivamente;
  Seja $\mathbf{x}_f$, $\mathbf{y}_f$, os vetores $\beta_f.x$, $\beta_f.y$, respectivamente;
  Seja $\theta \longleftarrow acos\left(\mathbf{x}_i \cdot \mathbf{x}_f\right)$;
  Seja $\mathbf{Rot}_1 \longleftarrow R\left(\theta, \mathbf{x}_i \times \mathbf{x}_f\right)$;
  se $\theta \neq 0$ **então**
    $\mathbf{y}_i \longleftarrow \mathbf{Rot}_1 \cdot \mathbf{y}_i$;
  fim
  Seja $\varphi \longleftarrow acos\left(\mathbf{y}_i \cdot \mathbf{y}_f\right)$;
  Seja $\mathbf{Rot}_2 \longleftarrow R\left(\varphi, \mathbf{x}_f\right)$;
  se $\varphi \neq 0$ **então**
    $\mathbf{y}_i \longleftarrow \mathbf{Rot}_2 \cdot \mathbf{y}_i$;
  fim
  se $\|\mathbf{y}_i - \mathbf{y}_f\| > 0$ **então**
    $\varphi \longleftarrow -\varphi$;
  fim
  Retorne φ;
fim

Algoritmo 5: Cálculo da matriz de rotação da interpolação entre β_i e β_f no instante t

Entrada: a base inicial, β_i, a tangente em t, $\mathbf{B}'(t)$, o ângulo φ em radianos e o instante t
Saída: matriz de rotação da interpolação no instante t

início
  Seja $\mathbf{x}_f$ o vetor $\mathbf{B}'(t)$;
  Seja $\mathbf{Rot}_1 \longleftarrow R\left(acos\left(\mathbf{x}_i \cdot \mathbf{x}_f\right), \mathbf{x}_i \times \mathbf{x}_f\right)$;
  Seja $\mathbf{Rot}_2 \longleftarrow R\left(t \cdot \varphi, \mathbf{x}_f\right)$;
  Retorne $\mathbf{Rot}_2 \cdot \mathbf{Rot}_1$;
fim

Por fim, basta apenas descrever o cálculo do Δt para completar o algoritmo de extrusão utilizado nesse trabalho. Como as curvas de Bézier, em sua maioria, não é linear, não se pode fazer $\Delta t = \frac{\Delta S}{resolução}$. Portanto, foi utilizado uma adaptação do algoritmo descrito em Hocevar, 2018 para o cálculo de Δt. Resumidamente, a ideia do algoritmo é que, para um Δt pequeno, pode-se aproximar $\Delta S = \|B'(t)\| \cdot \Delta t$, ou seja, se ΔS for pequeno, pode-se aproximar $\Delta t = \frac{\Delta S}{\|B'(t)\|}$. Caso ΔS não seja pequeno, o dividimos em n partes para obter distâncias menores e aproximar Δt na soma das aproximações menores de $\frac{\Delta S}{n}$, como mostrado no algoritmo 6.

Algoritmo 6: Cálculo do Δt da extrusão

Entrada: o instante t, a distância ΔS e a curva B
Saída: Δt

```
início
    Seja distLimite ← 0.01;
    se ΔS > distLimite então
        Seja n ← ⌈ΔS / distLimite⌉;
        Seja distMenor ← ΔS / n;
        Seja Δt ← 0;
        para i ← 1 até n por 1 faça
            Δt ← Δt + distMenor / ‖B'(t+Δt)‖;
        fim
        Retorne Δt;
    senão
        Retorne ΔS / ‖B'(t)‖;
    fim
fim
```

2.1.5 Triangulação dos vértices extrusados

A maioria dos modelos geométricos na computação gráfica são formados por diversos triângulos com vértices compartilhados, chamados de malhas triangulares, sendo responsáveis por representar a superfície do sólido, como dito por Marschner e Shirley (2016). Como o pipeline de renderização do Unity® utiliza triângulos, é necessário enviá-lo os triângulos formados pelos vértices que foram calculados pela extrusão através de um vetor de pontos, sendo que cada elemento do vetor representa um vértice e os elementos $3i$, $3i + 1$ e $3i + 2$, $i \in \mathbb{N}$ representam um triângulo. Para um triângulo ser renderizado no lado certo, os vértices devem estar no sentido horário. A figura 2.12 ilustra o vetor.

Portanto, os algoritmos de triangulação, dado os pontos extrusados, devem retornar um vetor de inteiros indicando os triângulos dos modelos. Como há dois tipos de modelos, os contínuos e os discretos ao longo do trajeto, existe dois modos de triangulá-los. Para os modelos discretos, o conjunto de vértices de cada instante é triangulado seguindo o próprio modelo do corte transversal, ignorando a existência dos outros instantes. Portanto, pode-se dizer que a extrusão apenas replicou o modelo ao longo da curva. A triangulação de um modelo discreto está exemplificada na figura 2.14.

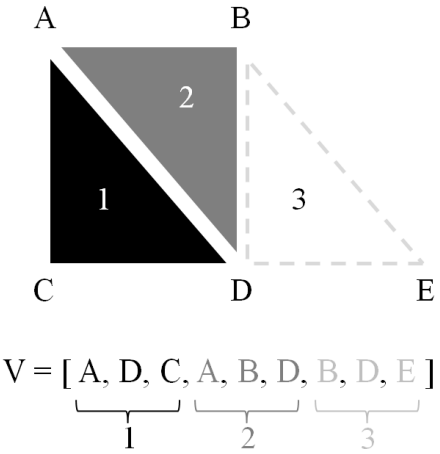


Figura 2.12: *Exemplo de um vetor, V, representando triângulos. V[0], V[1], V[2] representam o triângulo 1, que tem a orientação "correta"; V[3], V[4], V[5] representam o triângulo 2, que tem a orientação "correta"; V[6], V[7], V[8] representam o triângulo 3, que tem a orientação "incorreta".*

Seja *vértices*[][] o vetor de vetores que contêm os vértices extrusados em cada instante, ou seja, o elemento *vértices*[i][j] representa o vértice j do modelo no instante i da extrusão. Os modelos contínuos triangulam os vértice ligando os vértices do instante i aos do instante $i + 1$ de forma semelhante ao algoritmo 7 e ilustrada na figura 2.13. A triangulação de um modelo contínuo está exemplificado na figura 2.14.

Algoritmo 7: Simplificação do algoritmo de triangulação de um modelo contínuo

Entrada: o vetor de vetores que contêm os vértices extrusados *vértices*[][]
Saída: o vetor de inteiros indicando os triângulos

```
1  início
       /* Cálculo do tamanho do vetor de triângulos */
2      Seja t ⟵ vértices.Tamanho;
3      Seja v ⟵ vértices[0].Tamanho;
4      Seja triângulos[t][v];
       /* Para cada instante da extrusão, exceto o último */
5      para i ⟵ 0 até t − 1 por 1 faça
           /* Para cada vértice, exceto o último */
6          para j ⟵ 0 até v − 1 por 1 faça
               /* Calcule o índice do triângulo */
7              Seja índice ⟵ 6 · (i · t + j);
               /* Primeiro triângulo */
8              triângulos[índice] ⟵ (i) · v + (j);
9              triângulos[índice + 1] ⟵ (i + 1) · v + (j);
10             triângulos[índice + 2] ⟵ (i) · v + (j + 1);
               /* Segundo triângulo */
11             triângulos[índice + 3] ⟵ (i) · v + (j + 1);
12             triângulos[índice + 4] ⟵ (i + 1) · v + (j);
13             triângulos[índice + 5] ⟵ (i + 1) · v + (j + 1);
14         fim
15     fim
16     retorne triângulos;
17 fim
```

2.1.6 Junção de segmentos de trilho

No programa desenvolvido nesse trabalho, o segmento de trilho foi modelado como um objeto que armazena seus parâmetros de trajeto, como elevação; rotação; inclinação; comprimento e tipo de trilho, seu trajeto e suas malhas triangulares, sendo responsável por calculá-las. A fim de se construir uma montanha-russa, é preciso conectar segmentos de trilho um seguido do outro sem que o final de um segmento seja destoante do início do outro. Portanto, foi necessário definir a seguinte convenção ao construir a montanha-russa:

Convenção 1. *Sejam A e B segmentos de trilho tal que A seja seguido de B. Seja* $\mathbf{B}_A(t)$, $t \in [0,1]$ *e* $\mathbf{B}_B(t)$, $t \in [0,1]$ *os trajetos dos segmentos de A e B, respectivamente. Seja* β_A *a base final de A e* β_B *a base inicial de B. Logo, os segmentos devem satisfazer* $\mathbf{B}_A(1) = \mathbf{B}_B(0)$ *e* $\beta_A = \beta_B$.

Com a convenção 1, têm-se que o ponto inicial de um segmento é o mesmo do ponto final do anterior e que, tanto a tangente, quanto o vetor vertical do início do segmento são iguais ao do final do segmento anterior, logo, se um carro trafegará pelo trilho, ele não se ”teletransportará” e sua orientação será contínua.

Para satisfazer a convenção 1, bastou restringir a liberdade dos dois primeiros pontos

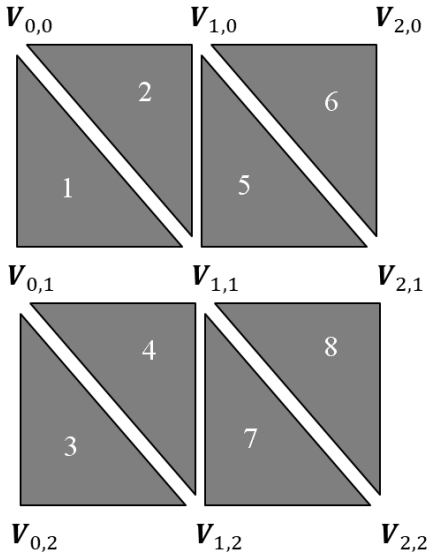


Figura 2.13: *Exemplo de triangulação de vértices de um modelo contínuo extrusado. Os vértices são* $\mathbf{V}_{i,j}$*, com* $i, j = 0, 1, 2$*, sendo que o primeiro índice do vértice representa a ordem do momento da extrusão e o segundo índice representa o índice original do modelo do corte transversal. Os triângulos estão em cinza e numerados.*

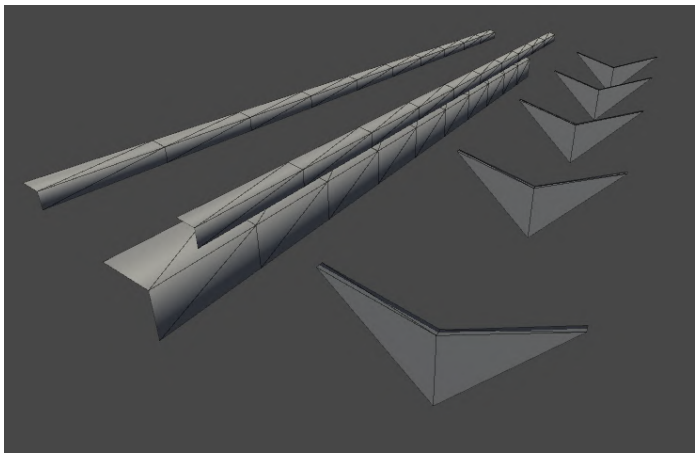


Figura 2.14: *Exemplo de triangulação de pontos extrusados. À esquerda está um modelo contínuo e à direita está um modelo discreto. As linhas representam as arestas e as faces representam os triângulos. As faces que não estão direcionadas à câmera não estão visíveis por causa do pipeline de renderização.*

de controle da curva de Bézier que representa o trajeto do segmento de trilho do seguinte modo: Seja A e B segmentos de trilho tal que A seja seguido de B. Seja $\mathbf{B}_A(t), t \in [0, 1]$ e $\mathbf{B}_B(t), t \in [0, 1]$ os trajetos dos segmentos de A e B, respectivamente. Seja $\mathbf{P}_{A_0}$, $\mathbf{P}_{A_1}$, $\mathbf{P}_{A_2}$, $\mathbf{P}_{A_3}$ os pontos de controle de $\mathbf{B}_A$ e $\mathbf{P}_{B_0}$, $\mathbf{P}_{B_1}$, $\mathbf{P}_{B_2}$, $\mathbf{P}_{B_3}$ os pontos de controle de $\mathbf{B}_B$. Portanto, $\mathbf{P}_{B_0} = \mathbf{P}_{A_3}$ e $\mathbf{P}_{B_1} = t \cdot \mathbf{B}'_A(1) + \mathbf{P}_{B_0}, t \in \mathbb{R}$. Cuidadosamente, o cálculo da Esfera Imaginária, descrita na subseção 2.1.3, realiza essas restrições.

2.1.7 Funções principais do Construtor

O Construtor exerce o trabalho principal no programa desenvolvido nesse trabalho, pois, é graças a ele que a montanha-russa do usuário é construída. Ele é responsável por criar, atualizar e deletar segmentos de trilhos, autocompletar a montanha-russa, checar intersecções, adicionar suporte aos trilhos, entre outras tarefas. Das informações armazenadas por ele, as principais são: lista ordenada dos segmentos de trilho da montanha-russa; base inicial, base final, posição inicial e posição final do último segmento de trilho; últimos parâmetros de trajeto de trilho (elevação, rotação, inclinação, comprimento); tipo do último trilho (plataforma, normal, alavanca ou freios).

Ao iniciar a construção, o Construtor recebe a posição inicial e base inicial da montanha-russa, os primeiros parâmetros de segmento de trilho e o tipo de montanha-russa e o tipo do primeiro trilho como parâmetro para realizar os processamentos necessários a fim de possibilitar a construção.

Para adicionar um trilho, o construtor calcula o trajeto, a posição final e a base final do novo segmento de trilho utilizando a Esfera Imaginária, descrita na subseção 2.1.3, utilizando a posição final e a base final do último segmento de trilho e os parâmetros de trilho informados pelo usuário anteriormente. Então, ele adiciona o novo segmento de trilho em sua lista na i-ésima posição, sendo que i é a quantidade atual de segmentos de trilho na montanha-russa, e armazena as informações de base inicial, base final, posição inicial e posição final do novo segmento de trilho.

Para atualizar o trajeto ou o tipo do último trilho, o Construtor apenas armazena os parâmetros recebidos, calcula o novo trajeto do último segmento de trilho caso necessário, atualiza o último segmento e atualiza as informações de base inicial, base final, posição inicial e posição final do último segmento.

Para deletar o último segmento de trilho, o construtor apenas remove o último segmento de sua lista e atualiza seus parâmetros armazenados sobre o novo último segmento de trilho.

Nas subseções seguintes, serão aprofundadas outras funções importantes do construtor.

2.1.8 Ferramenta de autocompletar trilho

Para auxiliar o jogador a “fechar” sua montanha-russa, foi desenvolvida uma ferramenta de autocompletar os trilhos, semelhante à ferramenta do Planet Coaster®. A ferramenta desenvolvida “conecta” o último segmento de trilho construído com a plataforma caso a tangente final do percurso do último segmento de trilho tenha um ângulo menor ou igual a $\frac{\pi}{4}$ em relação ao vetor entre a posição final do segmento e a posição inicial da plataforma e caso a distância com sinal do último ponto do último segmento de trilho em relação ao plano ortogonal à tangente inicial do percurso da plataforma seja menor do que um valor limite, ou seja, caso ele esteja ”atrás” do início da plataforma e caso tenha espaço para conectá-los. Seja $\mathbf{B}_i(t), t \in [0,1]$ o trajeto do último segmento de trilho construído e $\mathbf{B}_f(t), t \in [0,1]$ o trajeto da plataforma. Portanto, para verificar a primeira condição da ferramenta de autocompletar, basta verificar se $\mathbf{B}_i'(1) \cdot \left(\mathbf{B}_f(0) - \mathbf{B}_i(1)\right) \leq \frac{\pi}{4}$. Já para verificar

a segunda condição, foi utilizada a fórmula apresentada em Lengyel (2011) para calcular a distância com sinal, apresentada na equação 2.8.

$$dist = \left(\mathbf{B}'_f(0) \cdot \mathbf{B}_i(1)\right) - \left(\mathbf{B}'_f(0) \cdot \mathbf{B}_f(0)\right) \tag{2.8}$$

A ferramenta de autocompletar o trilho calcula o trajeto criando dois novos segmentos de trilho a fim de interpolar as orientações e posições. Sejam as trajetória desses dois segmentos $\mathbf{B}_a(t)$, $t \in [0,1]$ e $\mathbf{B}_b(t)$, $t \in [0,1]$, tal que $\mathbf{B}_i$ é seguida de $\mathbf{B}_a$, que, por sua vez, é seguida de $\mathbf{B}_b$ que, por sua vez, é seguida de $\mathbf{B}_f$. Logo, elas devem apresentam as seguintes propriedades:

- $\mathbf{B}_a(0) = \mathbf{B}_i(1)$;
- $\mathbf{B}_b(0) = \mathbf{B}_a(1)$;
- $\mathbf{B}_f(0) = \mathbf{B}_b(1)$;
- $\frac{\mathbf{B}'_a(0)}{\|\mathbf{B}'_a(0)\|} = \frac{\mathbf{B}'_i(1)}{\|\mathbf{B}'_i(1)\|}$;
- $\frac{\mathbf{B}'_b(0)}{\|\mathbf{B}'_b(0)\|} = \frac{\mathbf{B}'_a(1)}{\|\mathbf{B}'_a(1)\|}$;
- $\frac{\mathbf{B}'_f(0)}{\|\mathbf{B}'_f(0)\|} = \frac{\mathbf{B}'_b(1)}{\|\mathbf{B}'_b(1)\|}$.

Para facilitar os cálculos, foi definido que os raios das Esferas Imaginárias de $\mathbf{B}_a$ e $\mathbf{B}_b$ devem ser iguais. Seja $\mathbf{p}_i = \mathbf{B}_i(1)$, $\mathbf{p}_f = \mathbf{B}_f(0)$, $\mathbf{n}_i = \frac{\mathbf{B}'_i(1)}{\|\mathbf{B}'_i(1)\|}$, $\mathbf{n}_f = -\frac{\mathbf{B}'_f(0)}{\|\mathbf{B}'_f(0)\|}$ e seja r o raio das Esferas Imaginárias. A figura 2.15 ilustra o esquema desses pontos.

Para obter os trajetos, são necessários os raios das esferas, a elevação, a rotação e a inclinação dos trilhos. Primeiro, será calculado o raio das esferas. Pelas propriedades descritas anteriormente e pela figura 2.15, têm-se a relação 2.9.

$$2r = \left\| \left(\mathbf{p}_f + r \cdot \mathbf{n}_f\right) - \left(\mathbf{p}_i + r \cdot \mathbf{n}_i\right) \right\| \iff 2r = \left\| \left(\mathbf{p}_f - \mathbf{p}_i\right) + r \cdot \left(\mathbf{n}_f - \mathbf{n}_i\right) \right\| \tag{2.9}$$

Seja $\mathbf{p} = \mathbf{p}_f - \mathbf{p}_i$ e $\mathbf{n} = \mathbf{n}_f - \mathbf{n}_i$, então:

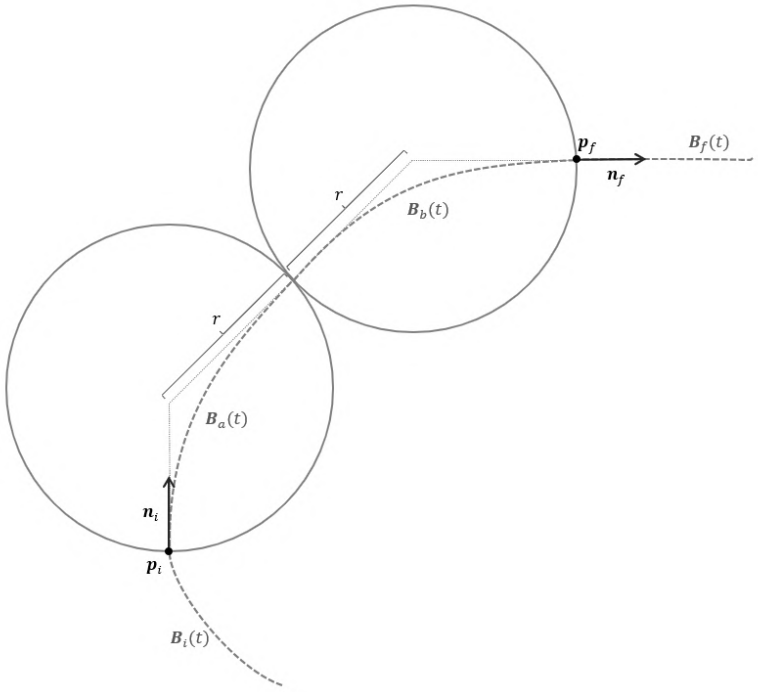


Figura 2.15: *Esquema dos trajetos* $\mathbf{B}_i(t)$ *e* $\mathbf{B}_f(t)$ *autocompletados por* $\mathbf{B}_a(t)$, $\mathbf{B}_b(t)$, $t \in [0,1]$, *sendo que* $\mathbf{p}_i = \mathbf{B}_i(1)$, $\mathbf{p}_f = \mathbf{B}_f(0)$, $\mathbf{n}_i = \frac{\mathbf{B}_i'(1)}{\|\mathbf{B}_i'(1)\|}$, $\mathbf{n}_f = -\frac{\mathbf{B}_f'(0)}{\|\mathbf{B}_f'(0)\|}$ *e* r *é o raio das Esferas Imaginárias de* $\mathbf{B}_a$ *e* $\mathbf{B}_b$.

$$
\begin{aligned}
&2r = \|\mathbf{p} + r \cdot \mathbf{n}\| \iff \\
&2r = \sqrt{\sum_{j=1}^{3} \left(p_j + r \cdot n_j\right)^2} \overset{r>0}{\Longrightarrow} \\
&4r^2 = \sum_{j=1}^{3} \left(p_j + r \cdot n_j\right)^2 \iff \\
&4r^2 = \sum_{j=1}^{3} \left(p_j^2 + r^2 n_j^2 + 2 r p_j n_j\right) \iff \\
&4r^2 = \sum_{j=1}^{3} \left(p_j^2\right) + r^2 \sum_{j=1}^{3} \left(n_j^2\right) + r\left(2 \sum_{j=1}^{3} \left(p_j n_j\right)\right) \iff \\
&4r^2 = \|\mathbf{p}\|^2 + r^2 \|\mathbf{n}\|^2 + r\,(2\mathbf{p} \cdot \mathbf{n}) \iff \\
&\left(4 - \|\mathbf{n}\|^2\right) r^2 + (-2\mathbf{p} \cdot \mathbf{n})\, r - \|\mathbf{p}\|^2 = 0
\end{aligned}
\tag{2.10}
$$

Portanto, basta resolver a equação de segundo grau em r. Seja $a = 4 - \|\mathbf{n}\|^2$, $b = -2\mathbf{p} \cdot \mathbf{n}$ e $c = -\|\mathbf{p}\|^2$. Como $r > 0$, então:

$$
r = \begin{cases} -\frac{c}{b}, & a = 0 \\ \frac{-b + \sqrt{b^2 - 4ac}}{2a}, & a \neq 0 \end{cases}
\tag{2.11}
$$

Com o raio das esferas calculado, é possível calcular os parâmetros de elevação, rotação e inclinação dos dois segmentos de trilho a serem construídos. Primeiro, são calculados os parâmetros de elevação e rotação do primeiro segmento, então calculada sua inclinação para então se obter a elevação, rotação e inclinação do segundo segmento. Para se calcular a rotação e a elevação de um trilho, deve-se ter a base inicial e a tangente final do trajeto.

Para o primeiro segmento, A, têm-se a base inicial, β, pois é a base final do último segmento construído. Já a tangente final, $\mathbf{t}_f$ é igual a $\frac{\mathbf{p}+r\cdot\mathbf{n}}{\|\mathbf{p}+r\cdot\mathbf{n}\|}$. O cáculo da elevação e da rotação do primeiro segmento é feito seguindo o algoritmo 8, sendo que "*TresRotações*(*base*, *elevação*, *rotação*, *inclinação*)" é o algoritmo 3 e "$ProjeteNoPlano(vetorAProjetar, vetorOrtogonalAoPlano)$" é uma função que retorna o vetor $vetorAProjetar$ projetado no plano definido por $vetorOrtogonalAoPlano$.

Resumidamente, o algoritmo 3 projeta, no plano definido pelo vetor $(0, 1, 0)$ (que é ortogonal ao plano), a tangente final e o vetor da base que será afetado pela rotação e que não é o eixo de "rotação" do trilho. Em seguida, os normaliza e calcula o ângulo com sinal do vetor projetado até a tangente final projetada. Esse ângulo calculado é o ângulo de "rotação", chamado de *rotação*, do segmento de trilho. Com o ângulo de "rotação" calculado, o algoritmo rotaciona a tangente final $-$*rotação* ao redor de $(0, 1, 0)$ e projeta o vetor $\beta.x$ e a tangente final rotacionada no plano definido pelo eixo de "elevação". Então, a elevação é o ângulo com sinal do vetor $\beta'.x$ à tangente rotacionada.

Algoritmo 8: Cálculo dos parâmetros de elevação e de rotação de um segmento de trilho

Entrada: a base inicial, β, e a tangente final, $\mathbf{t}_f$
Saída: os parâmetros de elevação e rotação, respectivamente

início
  Seja $\mathbf{v}_i \longleftarrow Normalize((\beta.x.x, 0, \beta.x.z))$;
  se $|\beta.x.y| == 1$ **então**
    $\mathbf{v}_i \longleftarrow Normalize((\beta.y.x, 0, \beta.y.z))$;
  fim
  Seja $\mathbf{v}_f \longleftarrow Normalize\left(\left(\mathbf{t}_f.x, 0, \mathbf{t}_f.z\right)\right)$;
  Seja $rotação \longleftarrow acos\left(\mathbf{v}_i \cdot \mathbf{v}_f\right)$;
  Seja $M_{tmp} \longleftarrow TresRotações(\beta, 0, rotação, 0)$;
  se $|\mathbf{v}_f - M_{tmp} \cdot \mathbf{v}_i| > 0$ **então**
    $rotação \longleftarrow -rotação$;
  fim
  Seja M_r a matriz de rotação de $-rotação$ radianos ao redor de $(0, 1, 0)$;
  $\mathbf{t}_f \longleftarrow M_r \cdot \mathbf{t}_f$;
  $\mathbf{v}_i \longleftarrow Normalize((\beta.x.x, 0, \beta.x.z))$;
  se $|\beta.y.y| < 0$ **então**
    $\mathbf{v}_i \longleftarrow Normalize((\beta.y.x, 0, \beta.y.z))$;
  fim
  Seja **cross** $\longleftarrow \mathbf{v}_i \times (0, 1, 0)$;
  se $\|\mathbf{cross}\| < 1$ **então**
    cross $\longleftarrow \beta.z$;
  fim
  Seja $\mathbf{p}_x \longleftarrow Normalize(ProjeteNoPlano(\beta.x, cross))$;
  Seja $\mathbf{p}_r \longleftarrow Normalize\left(ProjeteNoPlano\left(\mathbf{t}_f, cross\right)\right)$;
  Seja $elevação \longleftarrow acos(\mathbf{p}_x \cdot \mathbf{p}_r)$;
  $M_{tmp} \longleftarrow TresRotações(\beta, elevação, 0, 0)$;
  se $\|\mathbf{p}_r - M_{tmp} \cdot \mathbf{p}_x\| > 0$ **então**
    $elevação \longleftarrow -elevação$;
  fim
  retorne *elevação* e *rotação*;
fim

Para o cálculo dos parâmetros de elevação, rotação e inclinação dos segmentos de trilho, foi utilizado o algoritmo 9, sendo que $CalcEleRot(base, tangenteFinal)$ é uma chamada ao algoritmo 8. Os parâmetros de elevação e rotação são calculados diretamente usando $CalcEleRot(\beta_i, \mathbf{t})$. Então, a base inicial é rotacionada utilizando os parâmetros obtidos e rotacionada novamente, respeitando o algoritmo 3 para que $\beta'_i.x = \beta_f.x$. Com a base rotacionada, é medido o ângulo com sinal de $\beta'_i.y$ em direção a $\beta_f.y$ ao redor de $\beta_f.x$. Esse ângulo equivale a "inclinação" que a base inicial deve rotacionar para ser igual a base final, portanto, a inclinação do primeiro segmento equivale a metade desse ângulo.

Com os três parâmetros do primeiro segmento calculados, a base inicial é rotacionada os utilizando para se obter a base inicial do segundo segmento, logo, são calculadas a

elevação e a rotação do segundo segmento de trilho fazendo uma chamada a $CalcEleRot$ enviando a nova base e o vetor $-\mathbf{n}_f$, afinal, $-\mathbf{n}_f$ é a tangente final do segundo segmento. Por fim, é calculada a inclinação do segundo segmento rotacionando a base inicial do segundo segmento utilizando seus parâmetros e mensurando ângulo com sinal do novo $\beta_i'.y$ até $\beta_f.y$.

Algoritmo 9: Cálculo dos parâmetros de elevação, rotação e inclinação dos dois segmentos de trilho da ferramenta de autocompletar trilhos

Entrada: a base inicial, β_i, a base final, β_f, a posição final do último segmento de trilho, $\mathbf{p}_i$, posição inicial do primeiro segmento de trilho, $\mathbf{p}_f$, a tangente final, $\mathbf{t}_f$ e o raio, r, das novas Esferas Imaginárias

Saída: os parâmetros de elevação, rotação e inclinação, respectivamente, do primeiro e do segundo segmento de trilho, respectivamente

início

/* Calcula a tangente intermediária, $\mathbf{t}$ */

Seja $\mathbf{t} \longleftarrow Normalize\left(\left(\mathbf{p}_f + r \times \beta_f.x\right) - \left(\mathbf{p}_i + r \times \beta_i.x\right)\right)$;

Seja $\beta \longleftarrow \beta_i$;

/* Calcula a elevação e a rotação do primeiro segmento */

Seja $(e_1, r_1) \longleftarrow CalcEleRot\,(\beta, \mathbf{t})$;

/* Rotaciona a base para β para que $\beta.x = \beta_f.x$ */

$\beta \longleftarrow TresRotações(\beta, e_1, r_1, 0) \cdot \beta$;

Seja $(e_t, r_t) \longleftarrow CalcEleRot\left(\beta, \mathbf{t}_f\right)$;

$\beta \longleftarrow TresRotações(\beta, e_t, r_t, 0) \cdot \beta$;

/* Calcula a inclinação do primeiro segmento verificando o ângulo entre $\beta.y$ e $\beta_f.y$ */

Seja $i_1 \longleftarrow acos\left(\beta.y \cdot \beta_f.y\right)$;

/* Correção da direção da inclinação */

Seja $\mathbf{R}_i$ a matriz de rotação de i_1 radianos ao redor de $\beta_f.x$;

se $|\beta_f.y - \mathbf{R}_i \cdot \beta.y| > 0$ **então**

  $i_1 \longleftarrow -i_1$;

fim

/* Como há dois segmentos, a inclinação deve ser dividida por dois */

$i_1 \longleftarrow 0.5 \cdot i_1$;

/* Transforma β_i para se ter a base inicial do segundo segmento */

$\beta \longleftarrow TresRotações(\beta_i, e_1, r_1, i_1) \cdot \beta_i$;

/* Calcula a elevação e a rotação do segundo segmento */

Seja $(e_2, r_2) \longleftarrow CalcEleRot\left(\beta, \mathbf{t}_f\right)$;

$\beta \longleftarrow TresRotações(\beta, e_2, r_2, 0) \cdot \beta$;

/* Calcula a inclinação do segundo segmento verificando o ângulo entre o novo $\beta.y$ e $\beta_f.y$ */

Seja $i_2 \longleftarrow acos\left(\beta.y \cdot \beta_f.y\right)$;

/* Correção da direção da inclinação */

Seja $\mathbf{R}_i$ a matriz de rotação de i_2 radianos ao redor de $\beta_f.x$;

se $|\beta_f.y - \mathbf{R}_i \cdot \beta.y| > 0$ **então**

  $i_2 \longleftarrow -i_2$;

fim

retorne (e_1, r_1, i_1) e (e_2, r_2, i_2);

fim

Por fim, tendo calculado o raio das Esferas Imaginárias e os parâmetros de elevação, rotação e inclinação dos dois segmentos de trilho, é possível instanciá-los e completar o percurso da montanha-russa.

2.1.9 Checagem de intersecções entre segmentos de trilho

Para evitar que os segmentos de trilho não seguidos um do outro se cruzem, é preciso verificar se a menor distância entre suas trajetórias é maior do que o tamanho dos modelos de seus cortes ortogonais. Pelas suas trajetórias serem arcos ou retas, foi desenvolvido o algoritmo 10 que foi inspirado no algoritmo de busca binária. A fim de encontrar a menor distância entre as duas curvas B_1 e B_2, ele calcula os pontos $\mathbf{a}_1 = \leftarrow B_1(0); \mathbf{b}_1 = \leftarrow B_1(1); \mathbf{a}_2 = \leftarrow B_2(0); \mathbf{b}_2 = \leftarrow B_2(1)$ e as distâncias $d_1 = \|\mathbf{a}_1 - \mathbf{a}_2\|; d_2 = \|\mathbf{a}_1 - \mathbf{b}_2\|; d_3 = \|\mathbf{b}_1 - \mathbf{a}_2\|; d_4 = \|\mathbf{b}_1 - \mathbf{b}_2\|$. Então ele verifica a menor distância para repetir o processo com o "ponto médio" entre $\mathbf{a}_1$ e $\mathbf{b}_1$ e com o "ponto médio" entre $\mathbf{a}_2$ e $\mathbf{b}_2$ até a menor distância encontrada for menor do que uma distância limite ou até a distância entre os dois pontos calculados da mesma curva sejam pequenos o suficiente. Uma ilustração da execução do algoritmo está representada na figura 2.16.

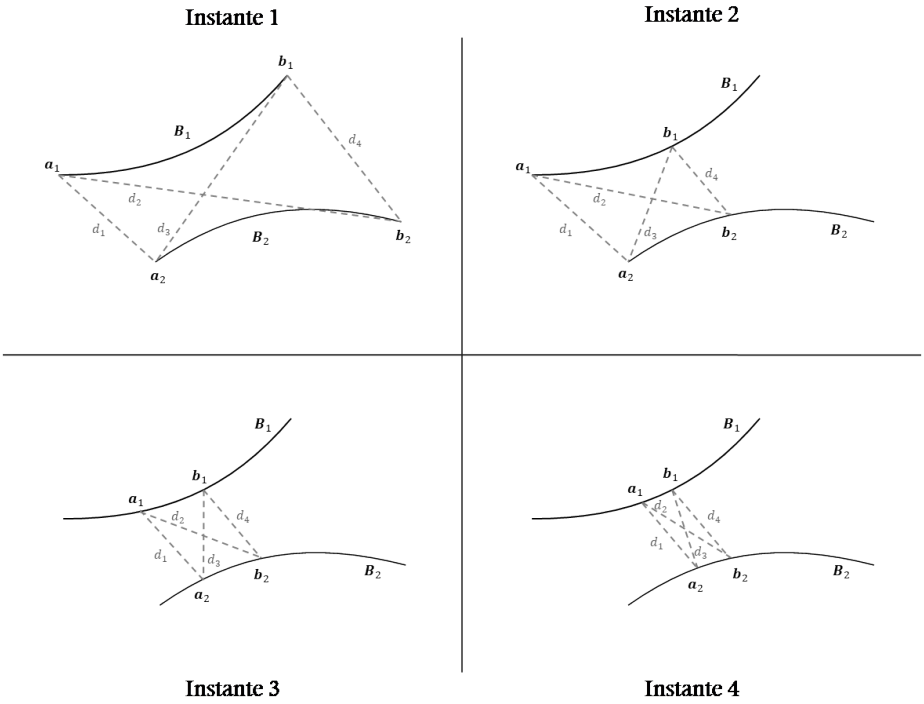


Figura 2.16: *Esquema de quatro instantes seguidos do algoritmo 10, Sendo* $\mathbf{B}_1$ *e* $\mathbf{B}_2$ *as curvas a serem verificadas se há intersecção. A distância* d_1 *é a menor das distâncias no primeiro instante,* d_4 *é a menor distância nos instantes dois e três.*

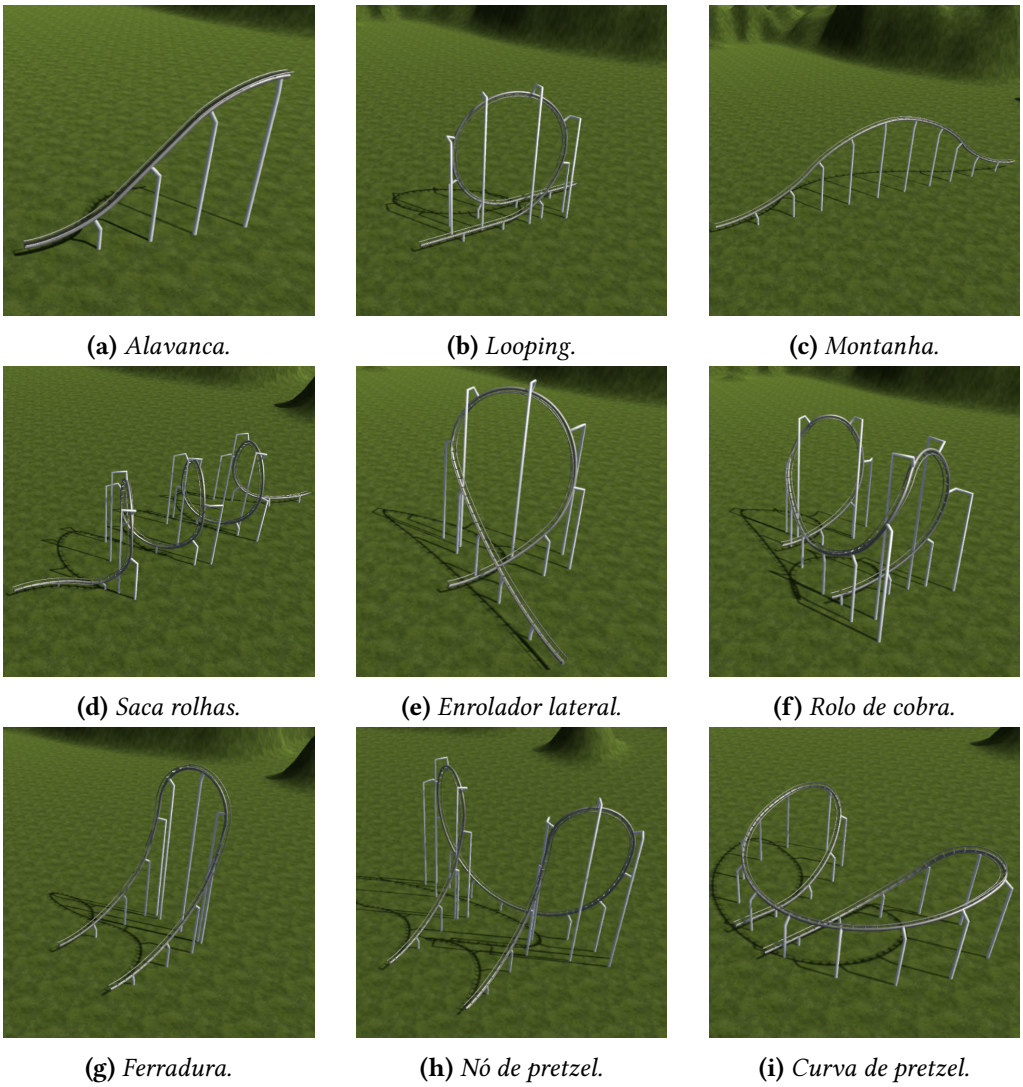
Algoritmo 10: Checagem se há intersecção entre dois segmentos de trilho não seguidos

Entrada: o trajeto do segmento 1, B_1, e o trajeto do segmento 2, B_2, sendo que ambos segmentos não são seguidos um do outro
Saída: um booleano indicando se houve intersecção

início
  Seja $distLimite \longleftarrow 2$;
  Sejam $t_{a_1} e t_{a_2}$ iguais a 0;
  Sejam $t_{b_1} e t_{b_2}$ iguais a 1;
  Seja $menorDist \longleftarrow \infty$;
  repita
    Seja $\mathbf{a}_1 \longleftarrow B_1(t_{a_1})$;
    Seja $\mathbf{a}_2 \longleftarrow B_2(t_{a_2})$;
    Seja $\mathbf{b}_1 \longleftarrow B_1(t_{b_1})$;
    Seja $\mathbf{b}_2 \longleftarrow B_2(t_{b_2})$;
    Seja $d_1 \longleftarrow \|\mathbf{a}_1 - \mathbf{a}_2\|$;
    Seja $d_2 \longleftarrow \|\mathbf{a}_1 - \mathbf{b}_2\|$;
    Seja $d_3 \longleftarrow \|\mathbf{b}_1 - \mathbf{a}_2\|$;
    Seja $d_4 \longleftarrow \|\mathbf{b}_1 - \mathbf{b}_2\|$;
    $menorDist \longleftarrow Mínimo(d_1, d_2, d_3, d_4)$;
    Seja $t_{m_1} \longleftarrow \frac{t_{a_1} + t_{b_1}}{2}$;
    Seja $t_{m_2} \longleftarrow \frac{t_{a_2} + t_{b_2}}{2}$;
    se $d_1 == menorDist$ **então**
      $t_{b_1} \longleftarrow t_{m_1}$;
      $t_{b_2} \longleftarrow t_{m_2}$;
    senão se $d_2 == menorDist$ **então**
      $t_{b_1} \longleftarrow t_{m_1}$;
      $t_{a_2} \longleftarrow t_{m_2}$;
    senão se $d_3 == menorDist$ **então**
      $t_{a_1} \longleftarrow t_{m_1}$;
      $t_{b_2} \longleftarrow t_{m_2}$;
    senão se $d_4 == menorDist$ **então**
      $t_{a_1} \longleftarrow t_{m_1}$;
      $t_{a_2} \longleftarrow t_{m_2}$;
    fim
    se $menorDist \leq distLimite$ **então**
      Retorne verdadeiro;
    fim
    se $\|\mathbf{a}_1 - \mathbf{b}_1\| \leq 0.1$ *e* $\|\mathbf{a}_2 - \mathbf{b}_2\| \leq 0.1$ **então**
      Retorne falso;
    fim
  até *verdadeiro*;
fim

2.1.10 Trechos pré-prontos

A fim de facilitar a construção da montanha-russa para o jogador, foram inseridos trechos pré-prontos de trilho comumente encontrados em montanhas-russas ao redor do mundo para que ele possa utilizá-los. Os trechos constituem de segmentos de trilho e foram parametrizados para que o usuário possa ajustá-los para melhor se encaixarem em suas montanhas-russas. Os trechos modelados nesse projeto foram baseados em alguns dos trechos encontrados em *List of roller coaster elements* s.d., sendo eles: looping; montanha; saca rolhas (corkscrew); enrolador lateral (sidewinder); rolo da cobra (cobra roll); ferradura (horseshoe); nó de pretzel (pretzel knot); curva de pretzel (pretzel curve). Também foram modelados alguns trechos adicionais, como alavanca; queda e curva. Alguns trechos pré-prontos modelados estão ilustrados na figura 2.17.



(a) *Alavanca.* **(b)** *Looping.* **(c)** *Montanha.*

(d) *Saca rolhas.* **(e)** *Enrolador lateral.* **(f)** *Rolo de cobra.*

(g) *Ferradura.* **(h)** *Nó de pretzel.* **(i)** *Curva de pretzel.*

Figura 2.17: *Trechos pré-prontos de trilhos modelados.*

Para adicionar um trecho pré-pronto, o Construtor deve adicionar segmentos de trilho seguidos com as propriedades descritas pelo trecho, por exemplo, ao inserir um trecho do tipo “alavanca” com n segmentos, o Construtor adiciona um segmento de trilho com

elevação local positiva, $n - 2$ segmentos com elevação local nula e um segmento com elevação local negativa, como ilustrado na figura 2.18, sendo que todos os segmentos de trilho adicionados são do tipo “alavanca”.

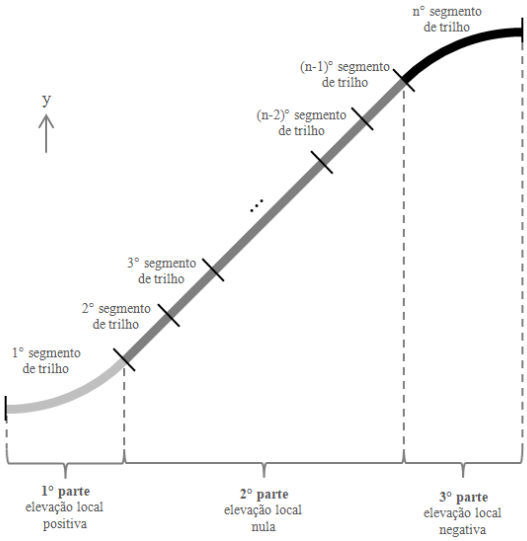


Figura 2.18: *Segmentos de trilho de um trecho pré-pronto do tipo “alavanca” com n segmentos e dividido em três partes. A primeira parte apresenta elevação local positiva com apenas um segmento, a segunda parte apresenta elevação local nula com n – 2 segmentos e a terceira parte apresenta elevação local negativa com apenas um segmento. Os tamanhos dos segmentos estão fora de escala.*

Todos os trechos de trilho pré-prontos modelados foram parametrizados, sendo que a maioria deles têm apenas a escala como parâmetro. Os trechos pré-prontos que apresentam rotação têm a orientação (direita ou esquerda) como parâmetro e os trechos como “alavanca” e “queda” permitem a alteração da altura, do ângulo de elevação e do ângulo de rotação, como ilustrado na figura 2.19.

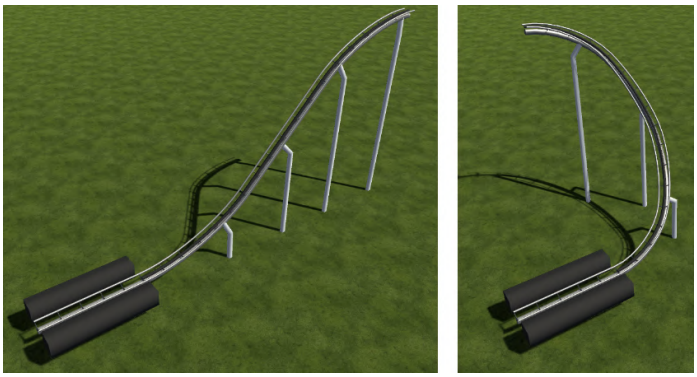
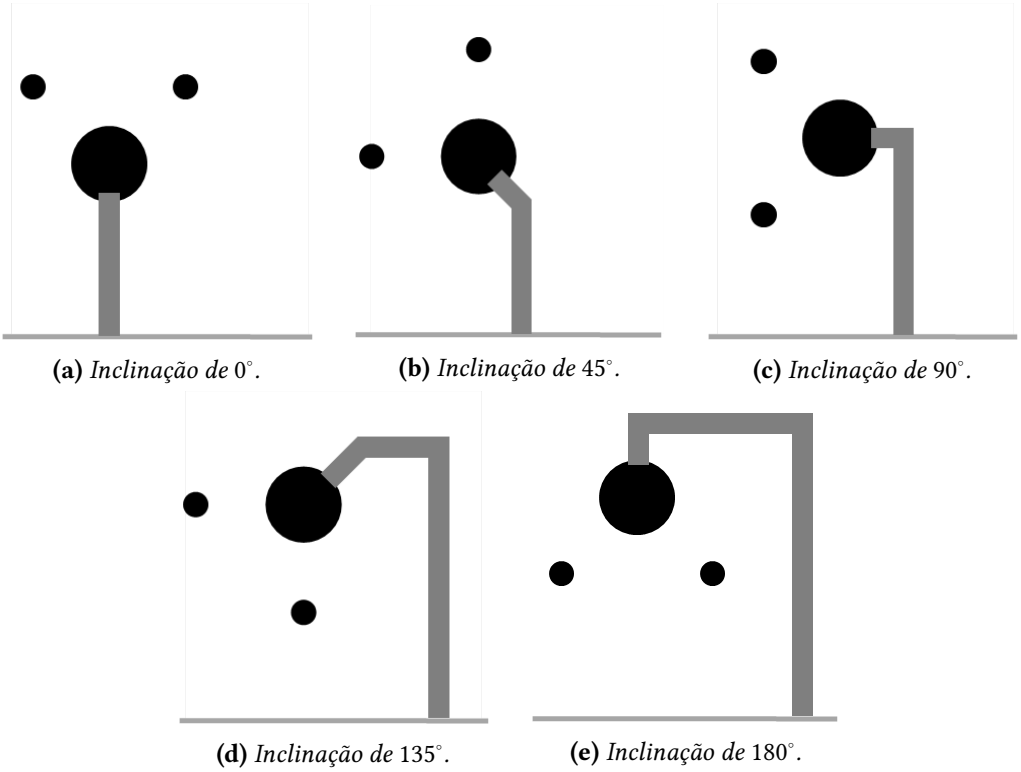


Figura 2.19: *Plataforma e trechos pré-prontos de trilho do tipo “alavanca” com parâmetros de rotação diferentes: à direita sem rotação e à esquerda com rotação.*

2.1.11 Suportes dos segmentos de trilho

Os suportes modelados dos trilhos são versões simplificadas e não fisicamente acurada dos suportes reais de trilhos de montanhas-russas, sendo puramente estéticos no programa

desenvolvido. Eles consistem, basicamente, de cilindros que conectam o trilho ao chão tendo três formatos diferentes de acordo com a inclinação deste. Um esquema dos formatos dos suportes está representado na figura 2.20 e um exemplo dos modelos do suporte estão presentes na figura 2.21.



(a) *Inclinação de* 0°. **(b)** *Inclinação de* 45°. **(c)** *Inclinação de* 90°.

(d) *Inclinação de* 135°. **(e)** *Inclinação de* 180°.

Figura 2.20: *Esquema do suporte de trilho de acordo com a inclinação do trilho. O trilho está representado em preto, o suporte em cinza escuro e o chão em cinza claro.*

2.2 Simulador

Para simular a velocidade dos vagões ao longo do trajeto da montanha-russa, foram modeladas quatro forças: gravidade; atrito; alavanca; freios. Como se quer saber a velocidade dos vagões, é necessário calcular a aceleração dessas forças. Por meio de experimentos, foi determinado que a aceleração da força da alavanca é igual a $6m/s^2$ e a aceleração da força de freios é igual a $-5m/s^2$. Já a aceleração da força de gravidade é calculada de acordo com a equação 2.12, sendo que β é a base do vagão.

$$a_g = 9.8 \cdot (0, -1, 0) \cdot \beta.x \tag{2.12}$$

A força de atrito tem um cálculo especial. Dada a aceleração do vagão já calculada aplicando as outras forças, a velocidade atual do vagão e a base do vagão, a aceleração do vagão, aplicando a força de atrito, é calculada seguindo o algoritmo 11, sendo que a função *Sinal* retorna o sinal do número. Esse calculo também foi modelado por meio de experimentos.

Figura 2.21: *Modelos dos suportes de trilho ao longo de um trilho com inclinação variável.*

Algoritmo 11: Cálculo da aceleração do vagão ao aplicar a força de atrito

Entrada: a aceleração do vagão já calculada aplicando as outras forças, a_{t+1} e a velocidade atual do vagão, v_t, e a base do vagão, β
Saída: a aceleração do vagão ao aplicar a força de atrito

início
  se $v_t > 0.07$ **então**
    $a_{t+1} \leftarrow a_{t+1} - Sinal(v_t) \cdot 0.004 \cdot Máximo(|v_t|, 10) \cdot (0.7 + 0.3 \cdot |(0,1,0) \cdot \beta.y|) \cdot 9.8$;
  fim
  senão
    $arrasto \leftarrow -Sinal(v_t) \cdot 0.04 \cdot (0.7 + 0.3 \cdot |(0,1,0) \cdot \beta.y|) \cdot 9.8$;
    se $|2 \cdot arrasto| > |a_{t+1}|$ **então**
      $a_{t+1} \leftarrow \frac{arrasto}{2}$;
    fim
    senão
      $a_{t+1} \leftarrow a_{t+1} + arrasto$;
    fim
  fim
  retorne a_{t+1};
fim

As forças serão aplicadas ou não de acordo com o tipo de trilho que o vagão está. No trilho do tipo "normal", são aplicadas apenas as forças de gravidade e de atrito. No trilho do tipo "freio", são aplicadas as forças de gravidade, de atrito e, se a velocidade do vagão for maior do que $4m/s$, é aplicada também a força de freio. No trilho do tipo "plataforma", é aplicada a força de gravidade, se a velocidade do vagão for maior do que $1m/s$, é aplicada a força de atrito e, se a velocidade do vagão for menor do que $4m/s$, é aplicada também a força de alavanca. Já no trilho do tipo "alavanca", é aplicada a força de gravidade, se a velocidade do vagão for maior do que $2m/s$, é aplicada a força de atrito e, se a velocidade do vagão for menor do que $6m/s$, é aplicada também a força de alavanca.

Para calcular a aceleração do vagão, basta somar as acelerações das forças aplicadas e,

caso for aplicável, aplicar a força de atrito. Para calcular a aceleração do carro (conjunto de vagões), é calculada a média das acelerações de seus vagões, pois foi considerado que, pelos vagões estarem presos uns aos outros, o carro é um corpo só.

Tendo a aceleração calculada, é preciso calcular a velocidade e mover o carro. Por apenas ser possível calcular a aceleração, velocidade e posição do carro a cada quadro do programa, não é possível ter um cálculo contínuo da velocidade, logo, ela foi aproximada utilizando o Método de Runge-Kutta de quarto grau pois esse método apresenta uma aproximação boa o suficiente para o propósito do jogo. Seja t_n o instante atual, t_{n-1} o instante anterior, Δt o intervalo entre os instantes t_{n-1} e t_n, v_{n-1} a velocidade do vagão no instante t_{n-1}, a_{n-1} a aceleração no instante t_{n-1} e seja $f(t_{n-1} + \Delta t, v_{n-1})$ a função que calcula a aceleração, a_{t_n} a partir de v_{n-1} e $t_{n-1} + \Delta t$. Logo, a velocidade no instante t_n é calculado de acordo com a equação 2.13.

$$
\begin{aligned}
k_1 &= f(t_{n-1}, v_{n-1}) \\
k_2 &= f(t_{n-1} + \frac{\Delta t}{2}, v_{n-1} + \frac{\Delta t}{2} k_1) \\
k_3 &= f(t_{n-1} + \frac{\Delta t}{2}, v_{n-1} + \frac{\Delta t}{2} k_2) \\
k_4 &= f(t_{n-1} + \Delta t, v_{n-1} + \Delta t \cdot k_3) \\
v_n &= v_{n-1} + \frac{\Delta t}{6} \cdot (k_1 + 2k_2 + 2k_3 + k4)
\end{aligned}
\tag{2.13}
$$

A posição do vagão é calculada ao longo dos percursos dos segmentos de trilho que são funções de Bézier. Portanto, para variar sua posição, é necessário variar o parâmetro das funções. Seja $B_i(p), p \in [0, 1]$ a função de Bézier que descreve o percurso do i-ésimo segmento de trilho. Logo, caso o vagão se locomova ΔS metros partindo de $B_i(p)$, é necessário calcular Δp, tal que $\Delta S = \int_p^{p+\Delta p} B_i(x)\,dx$ se $\Delta S >= 0$ e $\Delta S = \int_{p+\Delta p}^{p} B_i(x)\,dx$ se $\Delta S < 0$. Para realizar esse cálculo, foi utilizado o algoritmo 6, sendo que $\Delta S = v_n \cdot \Delta t$, o instante é p e a função é B_i. Caso $p + \Delta p$ seja maior do que 1, então é calculado o novo parâmetro p' na função do percurso do próximo segmento, $B_{(i+1)modr}(p), p \in [0, 1]$, sendo que r é a quantidade de segmentos de trilho. Caso $p + \Delta p$ seja menor do que 0, então é calculado o novo parâmetro p' na função do percurso do segmento anterior.

Portanto, a cada quadro da simulação, é calculada e atualizada a aceleração, a velocidade e a posição do carro no trilho da montanha-russa utilizando o intervalo de tempo entre o quadro anterior e o quadro atual como Δt. Esse método de cálculo permite que o carro seja pré-simulado nos trilhos, portanto, quando o jogador adiciona um segmento, o programa calcula a velocidade do carro ao longo do trajeto a partir da velocidade final do carro no segmento de trilho anterior e utilizando um Δt fixo. Com os trilhos pré-simulados, foi possível adicionar mapas de calor a eles indicando a velocidade ou a altura ou a força g do carro ao longo do percurso do segmento. Foi utilizado a paleta de cores “jet set” nos mapas de calor por ser fácil de visualizar a variação do valor ao longo da paleta. Um exemplo de mapa de calor dos segmentos de trilho está representado na figura 2.22.

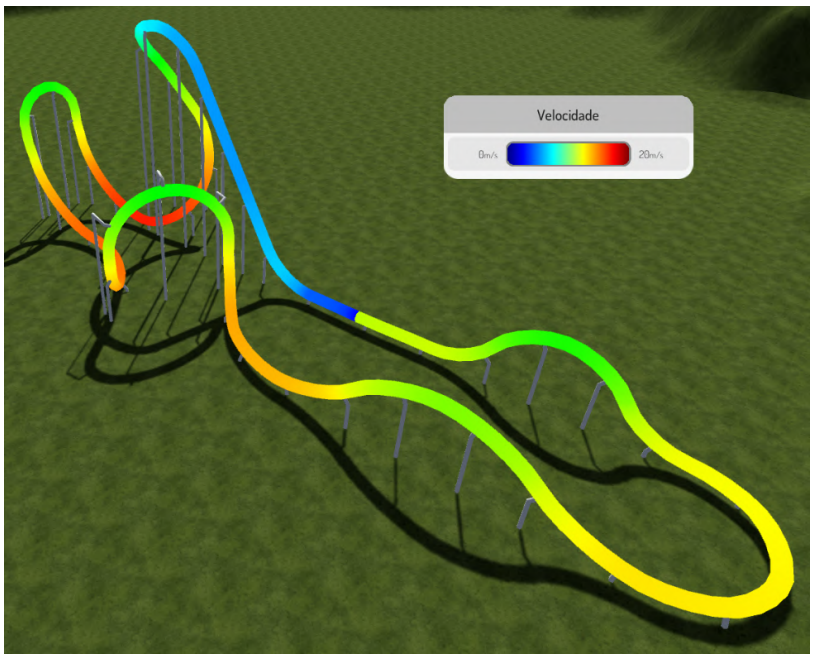


Figura 2.22: *Exemplo de um mapa de calor dos trilhos de uma montanha-russa virtual. O mapa de calor representa a velocidade do carro ao longo dos trilhos, sendo* $0m/s$ *azul escuro e* $20m/s$ *vermelho escuro.*

2.3 Gerador

Para gerar uma montanha-russa, o Gerador faz o uso dos trechos pré-prontos de trilho apresentados na subseção 2.1.10. Embora tenham sido apresentados na seção 2.1, eles foram desenvolvidos justamente para o funcionamento do Gerador, porém, ela foi adicionada como uma ferramenta do construtor para o usuário utilizar por facilitar a construção de suas montanhas-russas.

O Gerador foi baseado em uma gramática estocástica paramétrica desenvolvida nesse trabalho, porém, antes de apresentá-la, é preciso entender seu funcionamento. Para gerar uma montanha-russa que o trajeto não se intersecte, foram projetados cinco tipos de trajetos diferentes para o Gerador se basear. Cada trajeto é formado por um conjunto de percursos, que, por sua vez, são formados por conjuntos de trechos pré-prontos de trilhos. Cada percurso é de um tipo e podem ter restrições. Há cinco tipos de percursos: plataforma; reto; curva (curva de 90°); volta (curva de 180°); fim.

Os percursos do tipo “plataforma” e “fim” são percursos especiais. O percurso do tipo “plataforma” é formado por apenas um segmento de trilho reto do tipo “plataforma”, sendo que é o percurso inicial de todos os trajetos e não apresenta nenhuma restrição. O percurso do tipo “fim” é formado pelos segmentos de trilho gerados pela ferramenta de autocompletar os trilhos, sendo que também não apresenta nenhuma restrição e é o percurso final de todos os trajetos para que eles sejam cíclicos.

O percurso do tipo “reto” é formado por trechos pré-prontos de trilho que são, de certa forma, retos, como os trechos “montanha”, “looping”, “alavanca” (sem rotação) e ”saca-rolhas”. Cada percurso do tipo “reto” apresenta as restrições de comprimento mínimo

e comprimento máximo, ou seja, o tamanho do percurso gerado de uma ponta a outra deve satisfazer as restrições.

O percurso do tipo “curva” é formado por apenas um trecho pré-pronto de trilho que faz uma curva de 90°, como os trechos “curva”, “alavanca” (com rotação de 90°), “queda” (com rotação de 90°) e ”enrolador lateral”, sendo que apresenta apenas a restrição de orientação, ou seja, se a curva é para a direita ou para a esquerda.

O percurso do tipo “volta” é semelhante ao tipo “curva”, sendo formado por apenas um trecho pré-pronto de trilho que faz uma curva de 180°, como os trechos “curva”, “alavanca” (com rotação de 180°), “queda” (com rotação de 180°) e “ferradura”, sendo que apresenta apenas a restrição de orientação, ou seja, se a curva é para a direita ou para a esquerda.

Logo, um possível trajeto pode ser escrito como ”plataforma - curva à esquerda - curva à esquerda - reto por 20 metros - curva à esquerda - fim”, gerando um pequeno trajeto que se assemelha a uma elipse. Os trajetos traçados nesse trabalho são relativamente simples para que eles não se intersectem, sendo que três tem o formato semelhante à letra “O”, um à letra “T” e um à letra “U”.

Portanto, para gerar uma montanha-russa, o Gerador sorteia um dos cinco trajetos para seguir e, a cada percurso, gera os trechos pré-prontos de trilho do mesmo tipo do percurso respeitando as restrições e simula a velocidade do carro até o final do último segmento de trilho gerado. Para gerar os trechos pré-prontos, o gerador coleta quais são do tipo requisitado e quais deles aceitam a velocidade que o carro está para que ele consiga completar o percurso. Dos trechos coletados, ele sorteia um deles, seguindo a função de probabilidade que será comentada em 2.3.1, aleatoriza seus parâmetros e fixa os parâmetros necessários para seguir as restrições impostas, por exemplo, o percurso do tipo “curva” impõe que o parâmetro ”rotação” do trecho “curva” deve ser igual a 90°. Depois de aleatorizar os parâmetros, ele gera os segmentos, simula a velocidade do carro e, caso tenha completado o percurso, ele repete o algoritmo para o próximo percurso, caso contrário, ele coleta novamente os trechos pré-prontos possíveis e repete o resto do algoritmo até completar.

Para saber se o carro completa um determinado trecho pré-pronto de trilho dada a velocidade inicial dele no trecho, foi realizada uma série de simulações para cada parâmetro possível de todos os trechos pré-prontos. Então, para cada trecho, foi coletada a velocidade mínima que o carro precisou para completá-lo e foi modelada uma função matemática que, dada a velocidade inicial do carro, retorna seus parâmetros aleatorizados que tornem possível o carro completá-lo.

Para que o percurso do tipo “reto” tenha tamanho limitado, ao gerar um trecho pré-pronto de trilho, o gerador verifica se o tamanho ultrapassou o mínimo. Caso tenha ultrapassado o mínimo, ele verifica se ultrapassou o máximo e, se não ultrapassou, ele termina o percurso atual e processa o próximo e, se ultrapassou, ele deleta o trecho gerado, adiciona uma reta de um metro e gera novos trechos até gerar um trecho dentro do limite ou até a reta ultrapassar o máximo.

Com a explicação do funcionamento completo do Gerador feita, é possível apresentar a gramática no qual ele foi baseado.

2.3.1 Gramática

A gramática será escrita, por convenção, em inglês na seguinte formatação para diferenciá-la do texto:

```
The grammar will be written in this format.
```

Sintaxe

A sintaxe da gramática do gerador de montanhas-russas foi baseada nas sintaxes das gramáticas presentes no artigo HANAN, 1992 e na sintaxe da linguagem funcional SML, tendo estrutura semelhante a uma função de linguagem de programação.

Uma regra é composta pelo seu símbolo (nome), podendo apresentar um ou mais argumentos, uma condição para sua substituição (”execução”) ser válida e uma ou mais variáveis. Em seguida, necessariamente, apresenta a substituição e, por fim, pode ter seu peso na probabilidade acumulativa. A probabilidade da regra ser selecionada, supondo que ela é válida, é igual ao seu peso dividido pela soma dos pesos de todas as regras válidas.

```
symbol[(argument[, argument]*)] [: condition] =>
  [Let <variable> := <value>[,
  <variable> := <value>]* :]
    substitution
    [: probability]
```

Abreviações

A fim de simplificar a escrita e a leitura da gramática, serão incluídas algumas abreviações a ela. O primeiro conjunto de abreviações são referentes às propriedades locais do trilho, sendo elas a elevação, a rotação, a inclinação, o tamanho e o tipo do trilho. O tipo de trilho varia de zero a três, sendo 0 uma plataforma; 1 um trilho normal; 2 a alavanca e 3 o freio.

```
e => elevation
r => rotation
i => inclination
l => length
t => rail type
```

O segundo conjunto de abreviações são referentes às propriedades espacias do segmento de trilho, ou seja, sua posição global e sua base inicial.

```
p => position
b => basis
```

O terceiro conjunto de abreviações são referentes às propriedades físicas da simulação dos carros no segmento de trilho, sendo a velocidade, a força g, a altura do carro e a distância total entre o carro e a plataforma.

```
v => velocity
g => G Force
h => height
tl => total length
```

Por fim, a última abreviação é referente à matriz identidade.

```
I => identity matrix
```

Estruturas

Também a fim de simplificar a escrita e leitura da gramática, alguns valores são agrupados em estruturas. Os nomes das estruturas começam com "s" (de *struct*, do inglês, estrutura), exceto "rail" e "route". A primeira estrutura apresentada é referente às propriedades locais do segmento de trilho, chamada de "sr" (de *rail's structue*, do inglês, estrutura do trilho), contendo a elevação, rotação, inclinação e o tamanho do trilho. A soma de duas "sr" resulta na soma de todos os seus elementos, exceto o tamanho do trilho que é preservado o da estrutura à esquerda do operador de soma.

```
sr => (e,r,i,l)
sr + sr' => (sr.e + sr'.e, sr.r + sr'.r, sr.i + sr'.i, sr.l)
```

A segunda estrutura é referente às estruturas espaciais do segmento de trilho, chamada "ss" (de *spatial structure*, do inglês, estrutura espacial), contendo a posição inicial e a base inicial do trilho.

```
ss => (p, b)
```

A terceira estrutura é referente às propriedades físicas do carro no início do segmento de trilho, chamada "sp" (de *physics structure*, do inglês, estrutura física), contendo a velocidade, a força g, a altura do carro e a distância total entre o carro e a plataforma.

```
sp => (v, g, h, tl)
```

A quarta estrutura que é referente ao próprio segmento de trilho, chamada "rail" (do inglês, trilho). O segmento de contém as propriedades locais do segmento de trilho (sr), o tipo de trilho, as propriedades espaciais (ss) e as propriedades físicas (sp).

```
rail => (sr, t, ss, sp)
```

Por fim, a última estrutura é referente ao percurso, chamada de "route" (do inglês, rota). O percurso contém o seu tipo (plataforma, reta, curva, volta e final) e suas restrições. Para simplificar a gramática, foi suposto que o percurso "reto" não tem restrição e que tem apenas um trecho pré-pronto de trilho. A restrição da "curva" e da "volta" é um inteiro com o valor de "-1" representando a esquerda e "1" representando a direita.

```
route => (type, restriction)
```

Funções

A gramática apresenta algumas funções auxiliares para gerar a montanha-russa. A primeira, mais básica, é a "Instantiate" (do inglês, instanciar) que recebe um segmento de trilho como argumento e o instancia no jogo.

```
Instantiate(rail)
```

A função “Transf” (de *transform*, do inglês, transformar) recebe como argumento as propriedades locais e as propriedades espaciais de um segmento de trilho e retorna o cálculo das propriedades espaciais do final do trilho, ou seja, calcula a posição final e a base final do segmento de trilho.

```
Transf(sr, ss) => ss'
```

A função “Simulate” (do inglês, simule) recebe um segmento de trilho como argumento e retorna as propriedades físicas do carro no final do segmento de trilho.

```
Simulate(rail) => sp'
```

A função “Random” (do inglês, aleatório) recebe dois números e sorteia um número entre eles.

```
Random(a, b) => x ∈ [a, b[
```

A função “Autocomplete” (do inglês, autocompletar) instancia os dois segmentos de trilho gerados pela ferramenta de autocompletar.

```
Autocomplete() => x ∈ [a, b[
```

A função “GetPath” (do inglês, pegar caminho) retorna um dos cinco trajetos de montanha-russa em forma de lista de percursos sem o percurso inicial “plataforma”.

```
GetPath() => [route, route, route, ..., route]
```

Por fim, a função “NextProps” (do inglês, próximas propriedades) recebe um segmento de trilho e uma propriedade local de segmento de trilho e retorna uma tupla contendo a soma das propriedades locais enviadas e as propriedades locais do segmento de trilho, as propriedades espaciais no final do segmento de trilho e as propriedades físicas do carro no final do segmento de trilho. Ou seja, calcula as propriedades do próximo segmento de trilho. Também há uma sobrecarga da função que recebe apenas um segmento de trilho como argumento.

```
NextProps(rail, sr) => (sr + rail.sr, Transf(rail.sr, rail.ss), Simulate(
    rail))
NextProps(rail) => NextProps(rail, (0, 0, 0, 0))
```

Palavra Inicial

A palavra inicial da gramática é uma chamada à função “Plataform” dizendo, basicamente, que o segmento de trilho é reto, de tamanho igual a 5, com altura 1, sem rotação e que o carro, na posição inicial, tem velocidade igual a zero e lhe enviando os próximos percursos.

```
w0 => Plataform( (0, 0, 0, 5), ( (0, 1, 0), I ), (0, (0, 1, 0), 1, 0),
    GetPath(), null )
```

Gramática

A regra “Plataform” (do inglês, plataforma) recebe como argumentos as propriedades locais do segmento de trilho que será instanciado como plataforma, sua a posição e orientação, as propriedades físicas do carro no início do trilho, o trajeto e o percurso atual. Ela instancia a plataforma e faz uma chamada a “AddRoute” para adicionar os próximos percurso.

```
Plataform(sr, ss, sp, route::path, currentRoute) =>
  Let rail := (sr, 0, ss, sp) :
    Instantiate(rail)
    AddRoute(NextProps(rail, (0, 0, 0, 5)), path, route)
```

As regras “AddRoute” (do inglês, adicionar rota) basicamente filtra o tipo de percurso e faz uma chamada à regra responsável ao percurso.

```
AddRoute(sr,ss,sp, path, route, restriction) : (route.type == ‘‘plataform’’)
     => Plataform(sr, ss, sp, path, route) : 1
AddRoute(sr,ss,sp, path, route, restriction) : (route.type == ‘‘straight’’)
    => Straight(sr, ss, sp, path, route) : 1
AddRoute(sr,ss,sp, path, route, restriction) : (route.type == ‘‘curve’’) =>
    Curve(sr, ss, sp, path, route) : 1
AddRoute(sr,ss,sp, path, route, restriction) : (route.type == ‘‘turn’’) =>
    Turn(sr, ss, sp, path, route) : 1
AddRoute(sr,ss,sp, path, route, restriction) : (route.type == ‘‘end’’) =>
    Autocomplete() : 1
```

As regras “Straight” (do inglês, reta), “Curve” (do inglês, curva) e “Turn” (do inglês, volta) são as principais regras do gerador, pois são elas quem verificam quais tipos de trechos de trilho, como alavanca e inversões, são possíveis de adicionar dadas as propriedades físicas. Após verificar, elas sorteiam qual tipo de trecho de trilhos será adicionado em seguida e faz uma chamada à regra correspondente ao trecho sorteado.

```
Straight(sr, ss, sp, path, route) => StraightSegment(sr, ss, sp, path, route)
     : 0.1
Straight(sr, ss, sp, path, route) : (ss.h <= 4 && ss.v <= 6) => Lever(sr, ss,
     sp, path, route, 0) : 100
Straight(sr, ss, sp, path, route) : (ss.h > 10 && ss.v <= 6) => Fall(sr, ss,
     sp, path, route, 0) : 1000
Straight(sr, ss, sp, path, route) : (ss.h > 10 && ss.v > 6 && ss.v <= 10) =>
     Fall(sr, ss, sp, path, route, 0) : 1
Straight(sr, ss, sp, path, route) : (ss.h >= 5 && ss.h < 10 && ss.v <= 4) =>
     Fall(sr, ss, sp, path, route, 0) : 1
Straight(sr, ss, sp, path, route) : ss.v >= 15 => Looping(sr, ss, sp, path,
    route) : 1
Straight(sr, ss, sp, path, route) : ss.v >= 8 => Hill(sr, ss, sp, path,
    route) : 0.3
Straight(sr, ss, sp, path, route) : ss.v >= 17 => Corkscrew(sr, ss, sp, path,
     route) : 1

Curve(sr, ss, sp, path, route) : ss.v >= 3 => Curve(sr, ss, sp, path, route,
     90 * route.restriction) : 1
Curve(sr, ss, sp, path, route) : (ss.h <= 4 && ss.v <= 6) => Lever(sr, ss,
    sp, path, route, 30 * route.restriction, 3) : 100
Curve(sr, ss, sp, path, route) : (ss.h > 10 && ss.v <= 6) => Fall(sr, ss, sp,
     path, route, 30 * route.restriction, 3) : 1000
```

```
Curve(sr, ss, sp, path, route) : (ss.h > 10 && ss.v > 6 && ss.v <= 10) =>
    Fall(sr, ss, sp, path, route, 30 * route.restriction, 3) : 1
Curve(sr, ss, sp, path, route) : (ss.h >= 5 && ss.h < 10 && ss.v <= 4) =>
    Fall(sr, ss, sp, path, route, 30 * route.restriction, 3) : 1
Curve(sr, ss, sp, path, route) : ss.v >= 8 => Hill(sr, ss, sp, path, route,
    90 * route.restriction) : 0.3
Curve(sr, ss, sp, path, route) : ss.v >= 19 => Sidewinder(sr, ss, sp, path,
    route, route.restriction) : 1

Trun(sr, ss, sp, path, route) : ss.v >= 3 => Curve(sr, ss, sp, path, route,
    180 * route.restriction) : 1
Trun(sr, ss, sp, path, route) : (ss.h <= 4 && ss.v <= 6) => Lever(sr, ss, sp,
     path, route, 60 * route.restriction, 3) : 100
Trun(sr, ss, sp, path, route) : (ss.h > 10 && ss.v <= 6) => Fall(sr, ss, sp,
     path, route, 60 * route.restriction, 3) : 1000
Trun(sr, ss, sp, path, route) : (ss.h > 10 && ss.v > 6 && ss.v <= 10) =>
    Fall(sr, ss, sp, path, route, 60 * route.restriction, 3) : 1
Trun(sr, ss, sp, path, route) : (ss.h >= 5 && ss.h < 10 && ss.v <= 4) =>
    Fall(sr, ss, sp, path, route, 60 * route.restriction, 3) : 1
Trun(sr, ss, sp, path, route) : ss.v >= 8 => Hill(sr, ss, sp, path, route,
    180 * route.restriction) : 0.3
Trun(sr, ss, sp, path, route) : ss.v >= 21 => CobraRoll(sr, ss, sp, path,
    route, route.restriction) : 1
Trun(sr, ss, sp, path, route) : ss.v >= 17 => HorseShoe(sr, ss, sp, path,
    route, route.restriction) : 1
Trun(sr, ss, sp, path, route) : ss.v >= 19 => PretzelKnot(sr, ss, sp, path,
    route, route.restriction) : 1
Trun(sr, ss, sp, path, route) : ss.v >= 16 => PretzelCurve(sr, ss, sp, path,
     route, route.restriction) : 1
```

As regras “StraightSegment” (do inglês, segmento reto) e “CurveSegment” (do inglês, segmento curvo) são as regras mais simples. A primeira, instancia um segmento de trilho reto de tamanho igual a 1 metro e faz uma chamada a “AddRoute” para o próximo percurso ser gerado. O segundo, instancia uma curva com o ângulo igual ao mandado como argumento e faz uma chamada ao “AddRoute”.

```
StraightSegment(sr, ss, sp, route::path, currentRoute) =>
  Let rail := (sr, 0, ss, sp) :
    Instantiate(rail)
    AddRoute(NextProps(rail, (0, 0, 0, 1)), path, route)

CurveSegment(sr, ss, sp, route::path, currentRoute, rotation) =>
  Let rail := (sr, 0, ss, sp) :
    Instantiate(rail)
    AddRoute(NextProps(rail, (0, rotation, 0, 5)), path, route)
```

Antes de serem explicadas as regras referentes aos tipos de trechos de trilhos, será explicada a estrutura de um laço que será utilizado na gramática. O exemplo abaixo exibe como fazer um laço com o índice variando de zero a nove. Inicialmente, são chamadas as regras “Loop” (do inglês, ciclo) com os argumentos zero, representando o índice inicial, e dez, representando o número de iterações do laço. Como o índice é menor do que nove na primeira vez que “Loop” é chamado, será “executada” a regra de cima. No final da ”execução” da regra, é chamado novamente “Loop”, porém com o incremento de um no

índice. Após alguns ciclos, o índice será igual ao número de iterações, portanto, a regra que será "executada" será a debaixo, terminando o laço.

```
w_example => Loop(0, 10)

Loop(index, max) : index < max =>
    SomeRule()
    Loop(index + 1, max)

Loop(index, max) : index == max =>
    EndOfLoop()
```

O primeiro tipo de trecho de trilhos que será explorado é o "Lever" (do inglês, alavanca). A alavanca é o tipo de trecho de trilhos responsável por erguer o carro que está com baixa velocidade. A regra "Lever" basicamente aleatoriza seus argumentos e faz a chamada à regra que instancia seus parâmetros, que são: propriedades locais do segmento de trilho; propriedades espaciais; propriedades físicas; elevação; rotação (apenas presente no subtipo curvo); número de segmentos de trilho no trecho; trajeto; percurso atual.

```
Lever(sr, ss, sp, path, currentRoute, rotation, pieces) =>
    LeverInstantiate(sr, ss, 15 * Random(3, 6), rotation, pieces, path)
```

O trecho da alavanca é dividido basicamente em três partes: a primeira apresenta um segmento de trilho que possui elevação local positiva; a segunda apresenta a quantidade total de segmentos de trilho do trecho subtraída de dois segmentos de trilho que têm elevação local nula; a terceira apresenta apenas um segmento de trilho com a elevação local negativa, analogamente representado na figura 2.18.

A regra "LeverInstantiate" (do inglês, instanciar alavanca) instancia um segmento de trilho com elevação local positiva e, em seguida, faz uma chamada ao laço "LeverLoop".

```
LeverInstantiate(sr, ss, sp, e, r, n, path) =>
  Let rail := ((sr + (e, r, - r, 0)), 2, ss, sp) :
    Instantiate(rail)
    LeverLoop(NextProps(rail), e, r, 1, n, path)
```

O laço "LeverLoop" é responsável por instanciar os segmentos de trilho intermediários da alavanca e, no final do laço, instanciar o segmento de trilho com elevação local negativa e fazer uma chamada ao "AddRoute" para adicionar o próximo percurso.

```
LeverLoop(sr, ss, sp, e, r, j, n, path) : j < n - 1 =>
  Let rail := ((sr + (0, r, 0, 0), 2, ss, sp) :
    Instantiate(rail)
    LeverLoop(NextProps(rail), e, r, j + 1, n, path)

LeverLoop(sr, ss, sp, e, r, j, n, route::path) : j == n - 1 =>
  Let rail := ((sr - (e, -r, -r, 0), 2, ss, sp) :
    Instantiate(rail)
    AddRoute(NextProps(rail), path, route)
```

A regra "Fall" (do inglês, queda) tem funcionamento análogo à regra "Lever", porém o ângulo da elevação é negativo e o tipo de trilho é "normal", podendo ser escrita como:

```
Fall(sr, ss, sp, path, currentRoute, rotation, pieces) =>
    FallInstantiate(sr, ss, -15 * Random(3, 6), rotation, pieces, path)
```

```
FallInstantiate(sr, ss, sp, e, r, n, path) =>
  Let rail := ((sr + (e, r, - r, 0)), 1, ss, sp) :
    Instantiate(rail)
    FallLoop(NextProps(rail), e, r, 1, n, path)

FallLoop(sr, ss, sp, e, r, j, n, path) : j < n - 1 =>
  Let rail := ((sr + (0, r, 0, 0), 2, ss, sp) :
    Instantiate(rail)
    FallLoop(NextProps(rail), e, r, j + 1, n, path)

FallLoop(sr, ss, sp, e, r, j, n, route::path) : j == n - 1 =>
  Let rail := ((sr - (e, -r, -r, 0), 2, ss, sp) :
    Instantiate(rail)
    AddRoute(NextProps(rail), path, route)
```

Para simplificar a gramática, foi suposto que o trecho de trilhos pré-prontos do tipo "montanha" não tem parâmetros. A regra "Hill" (do inglês, montanha) cria uma lista com as propriedades dos segmentos de trilho que formam uma montanha e faz uma chamada a "Blueprint" (do inglês, projeto), o qual instancia todos os segmentos e, no final, faz uma chamada a "AddRoute". A fim de simplificar a gramática, os tamanhos dos segmentos da montanha têm cinco metros.

```
Hill(sr, ss, sp, path, currentRoute, rotation) =>
  Let localsrs :=
  [
    (60, rotation, -rotation, 5),
    (-60, rotation, 0, 5),
    (-60, rotation, 0, 5),
    (60, rotation, rotation, 5),
  ] :
    Blueprint(sr, ss, sp, path, localsrs)



Blueprint(sr, ss, sp, route::path, localsr:localsrs) : path != [] =>
  Let rail := ((sr + localsr, 1, ss, sp) :
    Instantiate(rail)
    Blueprint(NextProps(rail), path, localsrs)

Blueprint(sr, ss, sp, route::path, localsr) : path == [] =>
  Let rail := ((sr + localsr, 1, ss, sp) :
    Instantiate(rail)
    AddRoute(NextProps(rail), path, route)
```

As funções "Looping", "Corkscrew", "Sidewinder", "CobraRoll", "HorseShoe", "PretzelKnot" e "PretzelCurve" têm funcionamento análogo à regra "Hill", mas com a lista de propriedades do trilho de acordo com o formato do percurso de seu trecho pré-pronto, portanto, elas não serão apresentadas para simplificar o texto.

2.4 Terreno e Decoração

2.4.1 Terreno

O terreno foi desenvolvido utilizando uma técnica de mapa de altura comumente utilizada em desenvolvimento de jogos. A malha triangular do terreno, vista de cima, tem um formato quadrado formado por $n \times n$ vértices, tal que n é a quantidade de vértices de um lado, sendo que seus triângulos são formados pelos vértices $\mathbf{v}_{i,j}$, $\mathbf{v}_{i,j+1}$ e $\mathbf{v}_{i+1,j}$ e pelos vértices $\mathbf{v}_{i,j+1}$, $\mathbf{v}_{i+1,j+1}$ e $\mathbf{v}_{i+1,j}$, tal que $\mathbf{v}$ é um vértice e $i, j \in \mathbb{N}$ e $0 <= i, j < n - 1$. Os vértices são dispostos de tal maneira que o $\mathbf{v}_{i,j}$ está na posição $(i - \frac{n}{2}, y, j - \frac{n}{2})$, $y \in \mathbb{R}$, como esquematizado na figura 2.23.

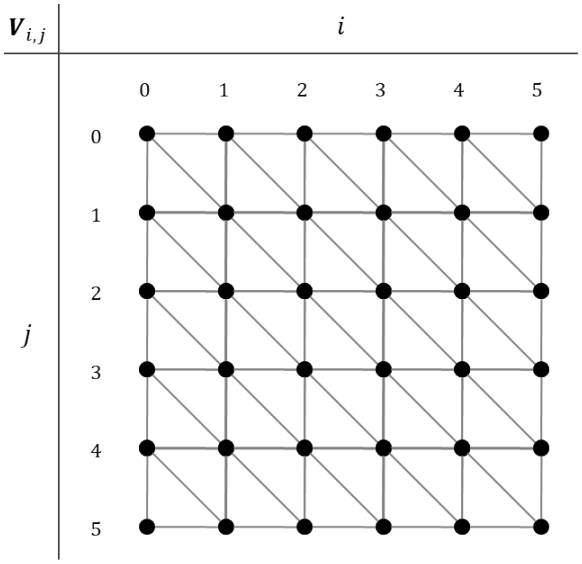


Figura 2.23: *Disposição e triangulação dos vértice da malha triangular do terreno.*

A altura de cada vértice é baseado em um mapa de altura, que é uma matriz de pontos flutuantes de tamanho $n \times n$. Para calcular a altura, é feita uma operação semelhante à convolução com um filtro gaussiano na matriz para suavizar a deformação da malha triangular, sendo que, para cada vértice, $\mathbf{v}_{i,j}$, sua altura é calculada multiplicando o valor de todos os vértices multiplicado por uma distribuição gaussiana centralizada em (i, j) amostrada na posição do vértice que está sendo multiplicado. Para otimizar o cálculo, são somadas apenas as multiplicações dos vértices que distam menos de 4 unidades do vértice que está sendo calculada a altura em relação aos índices. A malha resultante desse cálculo está presente na figura 2.24.

Por limitações do pipeline de renderização e por questões de otimização, a malha triangular do terreno foi dividida em 100 partes de tamanho $\frac{n}{10} \times \frac{n}{10}$, como ilustrado na figura 2.25.

Para o jogador deformar a malha, é projetado um raio da câmera ao cursor dele e o valor do elemento do mapa de altura com índice mais próximo ao índice do vértice da malha (desconsiderando as divisões) na posição (x, z) da intersecção do raio ao terreno será alterada, aumentando caso o jogador esteja elevando e diminuindo caso o jogador esteja rebaixando e, por fim, serão calculadas, novamente, as alturas dos vértices afetados por

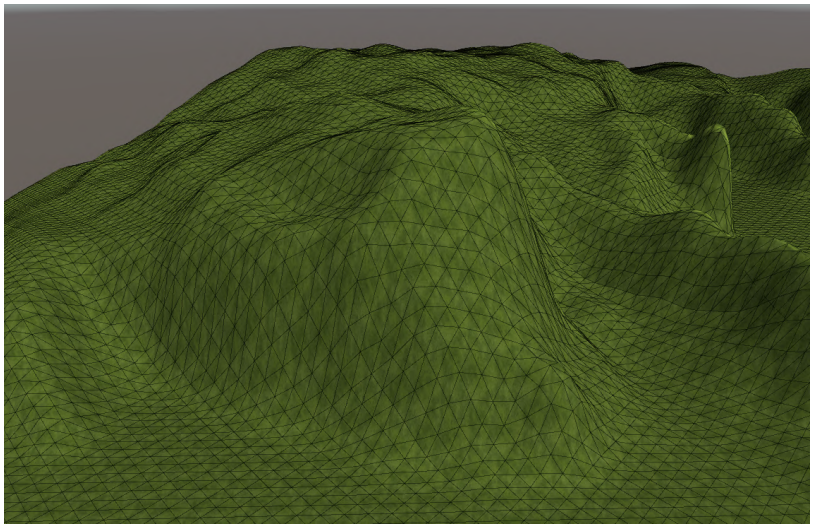


Figura 2.24: *Malha triangular do terreno. As linhas pretas representam as arestas dos triângulos da malha.*

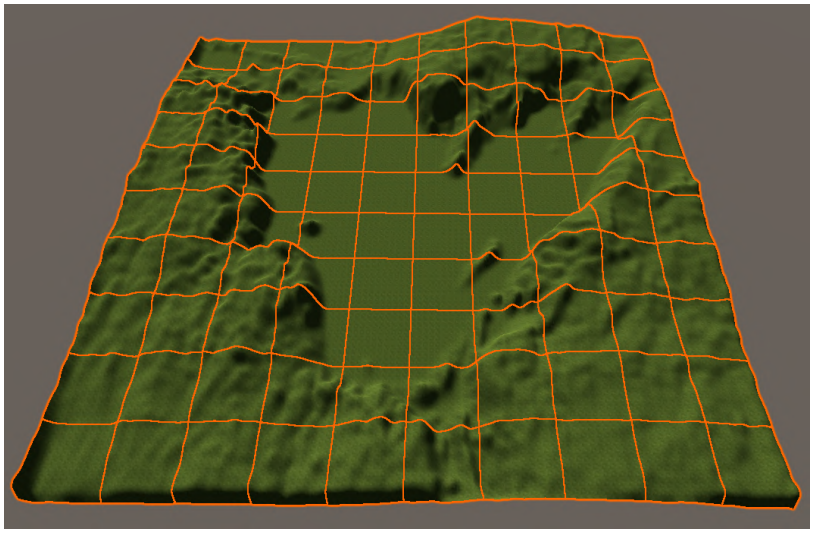


Figura 2.25: *Malha triangular do terreno dividida em 100 malhas triangulares de mesmo tamanho. As linhas laranjas representam a divisão.*

esse elemento alterado do mapa de altura. Por exemplo, o jogador está elevando o terreno e clica na posição $(0.1, 60, 5.3)$, como o vértice $\mathbf{v}_{i,j}$ é o mais próximo da posição clicada no plano (x, z), o valor elemento com índice (i, j) será aumentado e as alturas dos vértices com posição (x, y, z), tal que $-3 <= x <= 3$ e $2 <= z <= 8$, serão recalculadas.

2.4.2 Decoração

Para o usuário adicionar objetos no jogo, foi incluído o conjunto de modelos de domínio público disponível em https://www.kenney.nl/assets/castle-kit. Os modelos foram colocados em uma pasta especial do projeto para serem importados no jogo e instanciado pelo jogador. O usuário seleciona o objeto que deseja adicionar ao terreno e clica onde quer que seja colocado, então o programa projeta um raio da câmera ao cursor, checa

onde o raio intersectou com o terreno, arredonda a posição à grelha, instancia o objeto e o adiciona na lista de objetos instanciados para futura serialização. Caso o jogador queira deletar um objeto, basta ele clicar no objeto e deletá-lo, então o programa o deleta e o remove da lista de objetos instanciados.

2.5 Interface gráfica

Para criar o protótipo da interface gráfica, as funcionalidades que o usuário poderia interagir no jogo foram coletadas e organizadas em grupos, esclarecendo as diferentes partes da interface. Esse esquema está representado na figura 2.26.

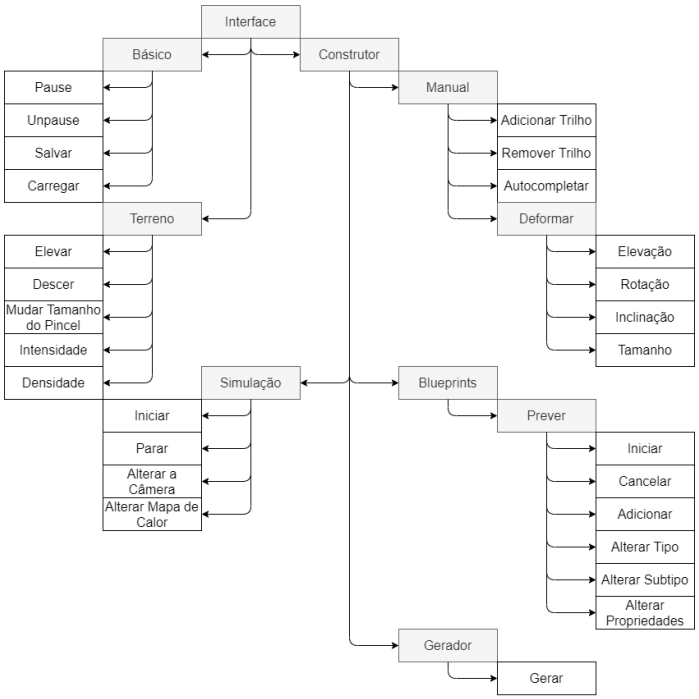


Figura 2.26: *Esquema das interações do usuário com o jogo.*

Para facilitar a construção da montanha-russa pelo jogador, foi prototipado um painel principal com todas as informações e botões necessários e de rápido acesso para a construção. Para o jogador alterar a elevação, rotação, inclinação e tamanho do segmento de trilho, foram colocadas setas com direções intuitivas no final do segmento de trilho que apontam para as direções de deformação. O protótipo está representado na figura 2.27.

Então foram prototipados os diferentes estados do painel principal para cada grupo de funcionalidades e foram indicadas as transições, que podem ser vistas na figura 2.28.

Após o término do protótipo da interface, ela foi desenvolvida no Unity® e submetida a testes por possíveis jogadores, os quais a avaliaram e deram sugestões para melhorá-la. Um dos testes submetidos foi a paleta de cores do painel principal, mostrado na figura 2.29.

Para que mais pessoas possam construir suas montanhas-russas virtuais, toda a interface foi traduzida para o inglês e português utilizando uma técnica comum em desenvolvi-

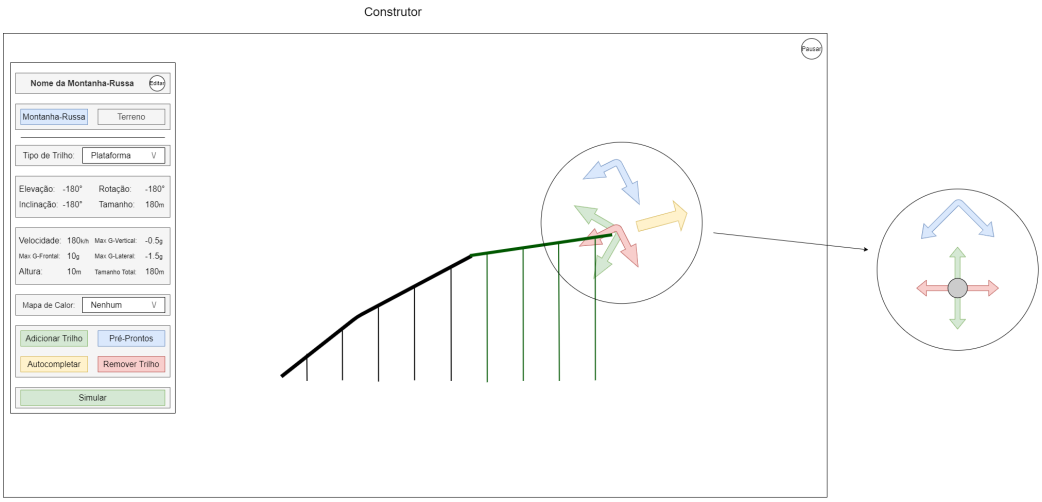


Figura 2.27: *Protótipo do construtor de montanhas-russas.*

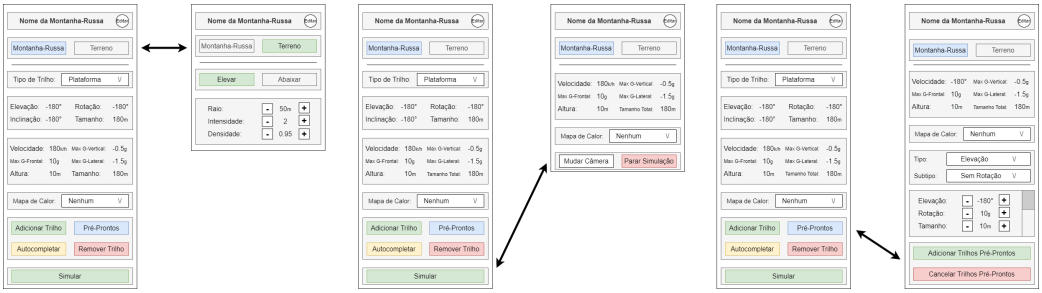


Figura 2.28: *Transições entre diferentes estados do painel principal.*

mento de jogos. Basicamente, o módulo de tradução do programa processa um arquivo de texto do idioma do usuário com todas as frases do jogo e suas respectivas “chaves”, então ele as armazena em um dicionário chave-valor e, quando aparece uma frase para jogador, a interface a requisita ao tradutor o enviando a chave correspondente. Para adicionar mais idiomas, bastaria criar um novo arquivo de texto em uma determinada pasta do projeto com as frases traduzidas e suas chaves originais e adicionar a opção do novo idioma no programa.

Nas subseções seguintes, serão exploradas algumas funcionalidades da interface.

2.5.1 Pré-visualização e deformação dos trilhos

A fim de facilitar a construção das montanhas-russas pelo usuário, foi criada uma funcionalidade de pré-visualização do próximo trilho a ser adicionado. Resumidamente, o Construtor instancia, no final do último segmento de trilho construído, um novo segmento de trilho que é renderizado com transparência verde caso não haja intersecção e vermelho caso contrário, mostrando como ele ficaria e indicando se o jogador pode ou não construir com os parâmetros adotados.

Para melhorar a experiência do jogador ao manipular os parâmetros do segmento de trilho que está sendo construído, foram incluídas setas no final do segmento de trilho que está no modo de prévia que indicam a direção da deformação do trilho se o usuário

Figura 2.29: *Diferentes versões de cores do painel principal do construtor de montanhas-russas. O painel da esquerda foi considerado o melhor pela maioria dos possíveis jogadores entrevistados informalmente.*

arrastá-las para essa direção. As setas podem ser vistas na figura 2.30, sendo que a seta verde altera a elevação do segmento de trilho, a vermelha altera a rotação, a azul altera a inclinação e a amarela altera o tamanho. Quando o usuário clica, é projetado um raio da câmera ao cursor do jogador e é verificado se o raio intersecta com alguma delas, então, caso tenha intersectado, é guardada a posição do cursor na tela. A cada quadro após o usuário ter clicado e até ele soltar, é calculado o deslocamento do cursor e é projetado na direção da seta projetada na tela, assim, é calculada a variação do parâmetro que está sendo alterado, o atualiza e atualiza o segmento de trilho que está em modo de prévia.

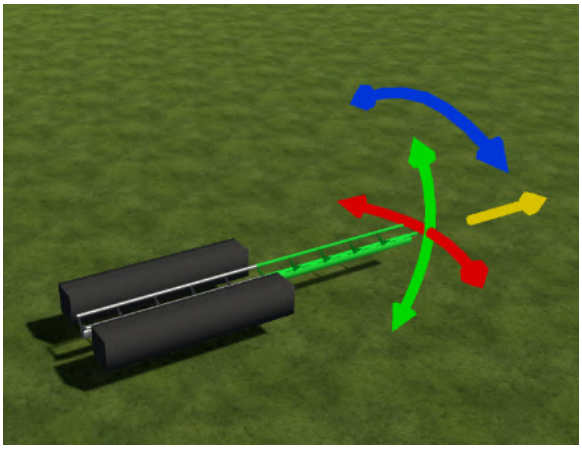


Figura 2.30: *Setas de suporte à construção da montanha-russa virtual pelo usuário. A seta verde altera a elevação do segmento de trilho, a vermelha altera a rotação, a azul altera a inclinação e a amarela altera o tamanho.*

2.5.2 Decoração

Quando o jogador deseja adicionar um objeto de decoração em seu terreno, ele precisa selecioná-lo entre outros objetos na interface, como na figura 2.31. Para que ele possa selecionar, é preciso que os objetos tridimensionais sejam instanciados na cena, renderizados em um botão bidimensional e não podem aparecer "flutuando" no terreno. Portanto, foram criadas câmeras especiais na cena que renderizam apenas esses objetos e a imagem resultante de sua renderização é colocada no botão. Para evitar que a câmera principal não mostrasse esses objetos flutuando, eles foram inseridos em uma nova camada que não é renderizada por essa câmera em específica.

Figura 2.31: *Painel de seleção de objetos de decoração.*

2.5.3 Serialização

Para que o jogador possa retornar à sua montanha-russa, foi desenvolvida a ferramenta de salvar e carregar a montanha-russa e o cenário que serializa e desserializa as modificações que o usuário fez no jogo. A ferramenta salva as informações em um arquivo binário junto com uma foto do cenário tirada pelo jogador em uma pasta com o nome que ele deu à sua montanha-russa. O arquivo binário tem o seguinte formato, sendo que os subitens são estruturas que se repetem:

- Versão do modo de salvar (4 bytes);
- Bandeiras gerais (4 bytes);
- Identificador do modelo do trilho (4 bytes);
- Quantidade de segmentos trilhos (4 bytes);
  - Elevação do segmento (4 bytes);
  - Rotação do segmento (4 bytes);

  - Inclinação do segmento (4 bytes);
  - Comprimento do segmento (4 bytes);
  - Tipo do trilho (4 bytes);
  - Bandeiras do segmento (4 bytes);
- Quantidade de objetos decorativos (4 bytes);
  - Quantidade de bytes do nome do objeto (4 bytes);
  - Nome do objeto (número variável de bytes);
  - Posição (x, y, z) do objeto (12 bytes);
  - Orientação do objeto (4 bytes);
- Tamanho do mapa de altura do terreno (4 bytes);
  - Valor do elemento do mapa de altura (4 bytes);

As ”bandeiras gerais” armazenam se foram salvos os objetos decorativos e os terrenos no arquivo, tendo espaço para armazenar mais 22 informações. Já as bandeiras dos trilhos armazenam a informação de se o segmento de trilho foi construído pela ferramenta de autocompletar para que, ao carregá-lo, ele seja construído pela própria ferramenta.

Capítulo 3

Resultados

O “RollerCoasterGenerator” permite que o usuário construa sua própria montanha-russa colocando e ajustando os trilhos manualmente, como na figura 3.1, com trechos pré-prontos de trilho, como na figura 3.2. Ao final da construção, o jogador pode utilizar a ferramenta de autocompletar os trilhos. Para auxiliar o usuário a construir sua montanha-russa, foi desenvolvida uma ferramenta que exibe o mapa de calor da velocidade do carro ao longo do trilho, como nas figuras 3.3 e 3.4. Com essa ferramenta, é possível verificar se o carro ultrapassa o segmento de trilho ou o trecho pré-pronto de trilho que está sendo construído.

O usuário também pode simular o carro ao longo do trilho, como na figura 3.5, vendo sua velocidade, força g e altura a cada instante. O jogador também pode ter a visão do passageiro no carro, como na figura 3.6. O usuário também pode alterar o relevo do terreno, como na figura 3.7, e colocar objetos de decoração, como na figura 3.8. Com a montanha-russa construída, o relevo do terreno alterado e com as decorações colocadas, o usuário pode "dar uma volta"em sua montanha russa, como na figura 3.9.

O usuário pode sempre salvar sua montanha-russa junto com as decorações colocadas, as alterações realizadas no relevo do terreno e uma foto tirada, como na figura 3.10. O usuário também pode sempre carregar uma montanha-russa salva anteriormente, como mostrado na figura 3.11. Toda a interface do programa foi traduzida para português e inglês. Além de construir sua montanha-russa, o jogador pode gerá-la. Alguns exemplos de montanhas-russas geradas estão representadas nas figuras 3.12 a 3.21.

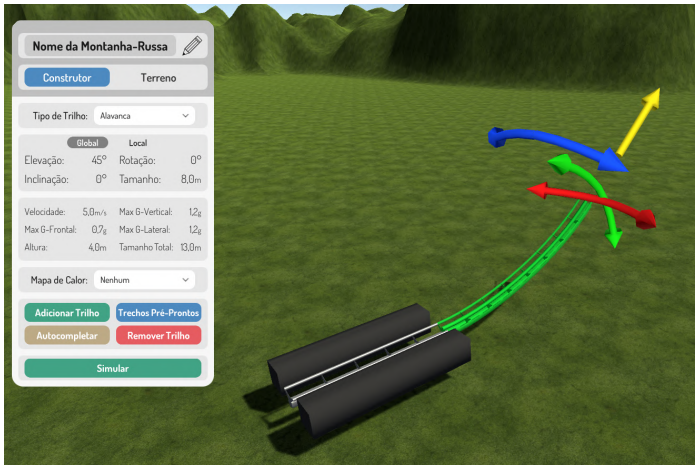


Figura 3.1: *Construtor de montanhas-russas do programa desenvolvido nesse trabalho.*

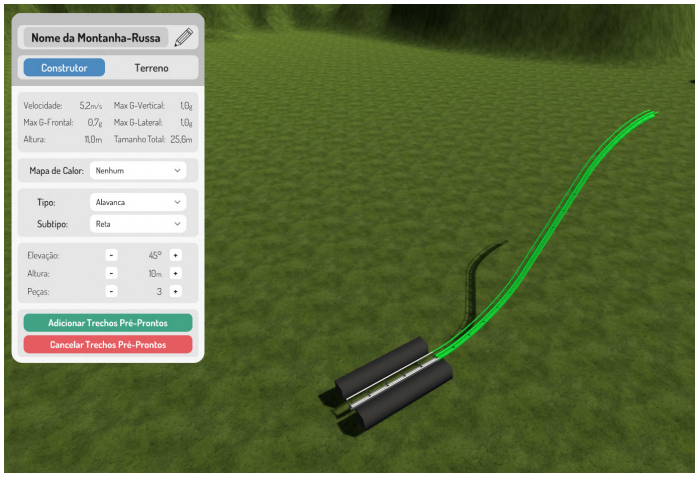


Figura 3.2: *Utilização de trechos pré-prontos de trilho no programa desenvolvido nesse trabalho.*

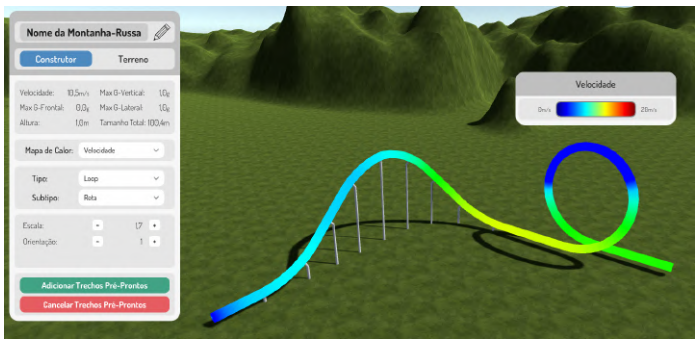


Figura 3.3: *Mapa de calor da velocidade do carro ao longo de uma montanha-russa sendo construída no programa desenvolvido nesse trabalho. O carro não conseguiria completar o looping.*

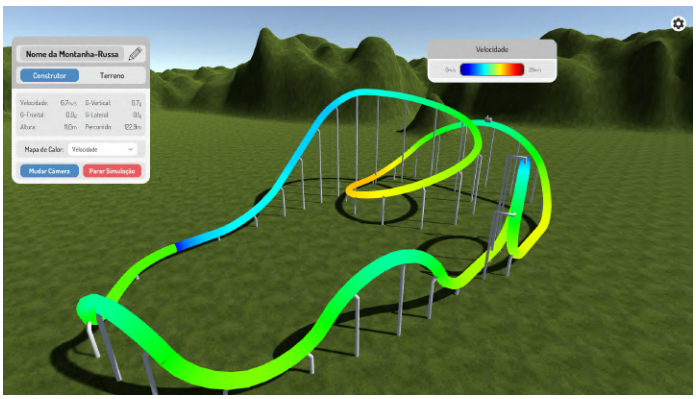


Figura 3.4: *Mapa de calor da velocidade do carro ao longo de uma montanha-russa construída no programa desenvolvido nesse trabalho.*

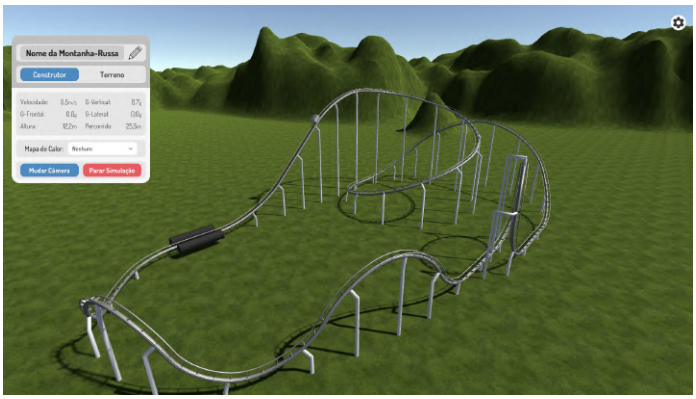


Figura 3.5: *Simulação do carro ao longo da montanha-russa construída no programa desenvolvido nesse trabalho.*

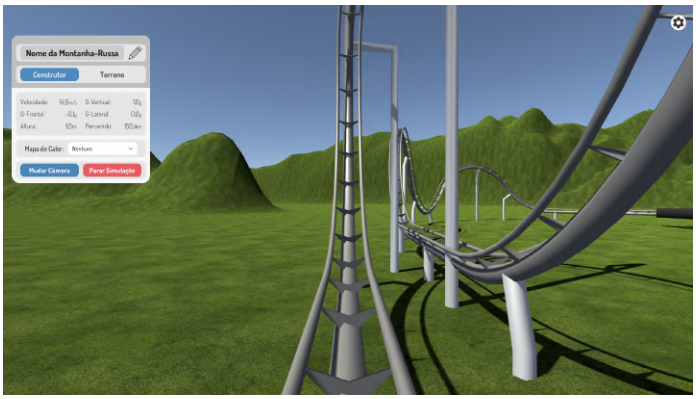


Figura 3.6: *Visão do passageiro no carro ao longo da montanha-russa construída no programa desenvolvido nesse trabalho.*

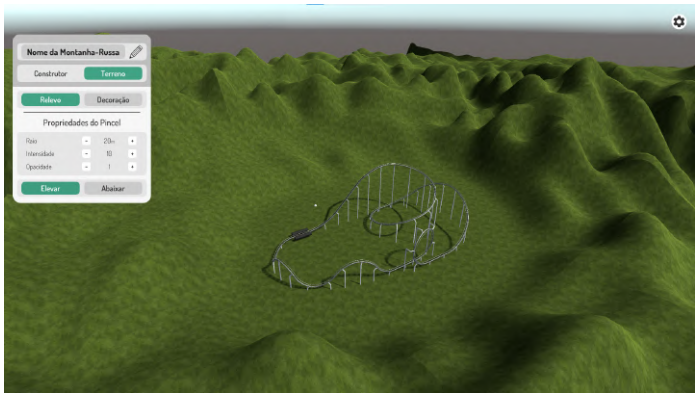


Figura 3.7: *Alteração do relevo do terreno do programa desenvolvido nesse projeto.*

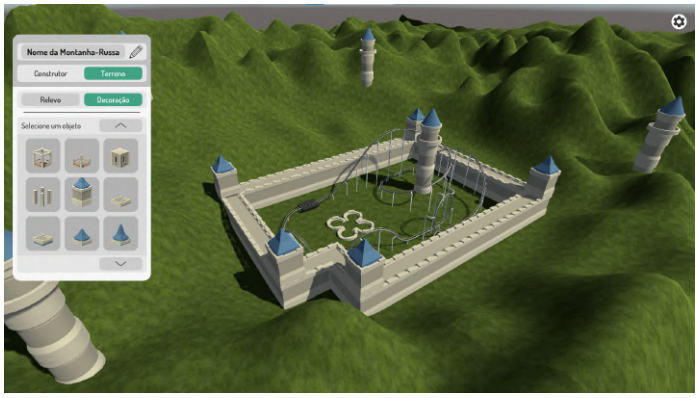
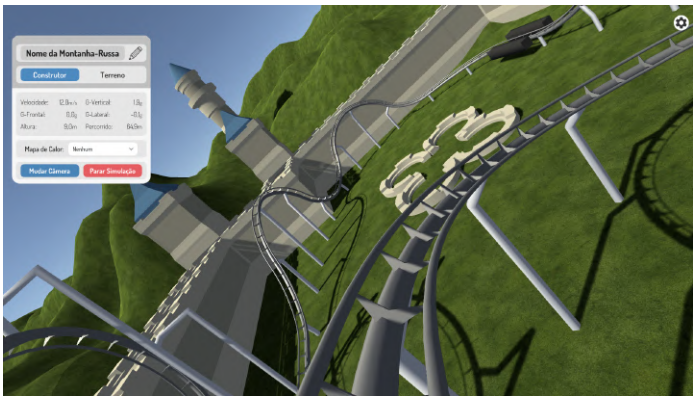
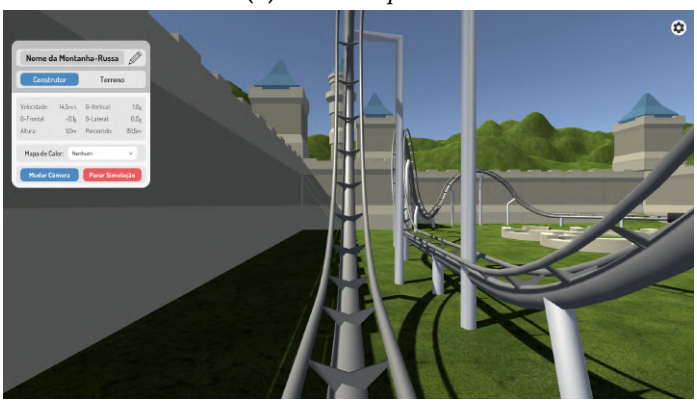


Figura 3.8: *Decoração do terreno do programa desenvolvido nesse projeto.*

(a) *Saindo da estação.*



(b) *Primeira queda.*



(c) *Entrando em um looping.*

Figura 3.9: *Visão do passageiro, em diferentes momentos, no carro da montanha-russa construída no programa desenvolvido nesse projeto.*

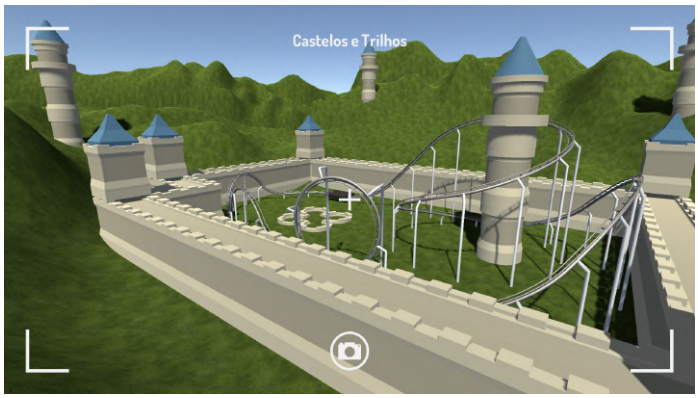


Figura 3.10: *Ferramenta de tirar foto para salvar a montanha-russa construída pelo usuário no programa desenvolvido nesse projeto.*

Figura 3.11: *Painel de carregar montanhas-russas salvas anteriormente pelo usuário no programa desenvolvido nesse projeto.*

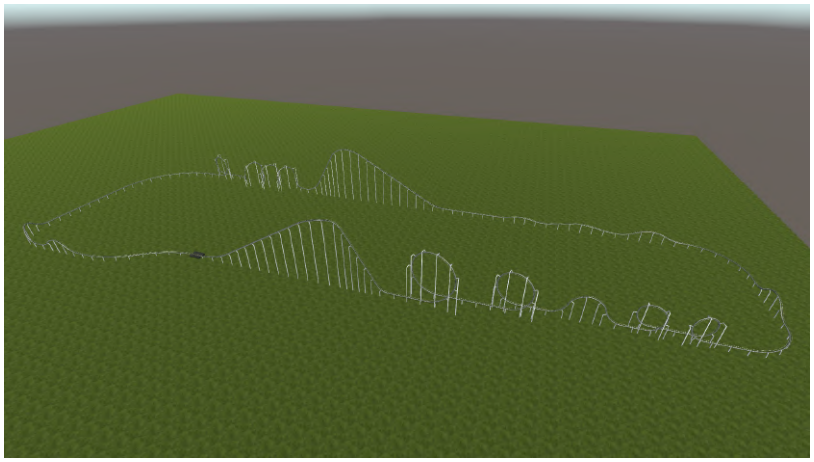


Figura 3.12: *Exemplo #1 de montanha-russa gerada no programa desenvolvido nesse projeto.*

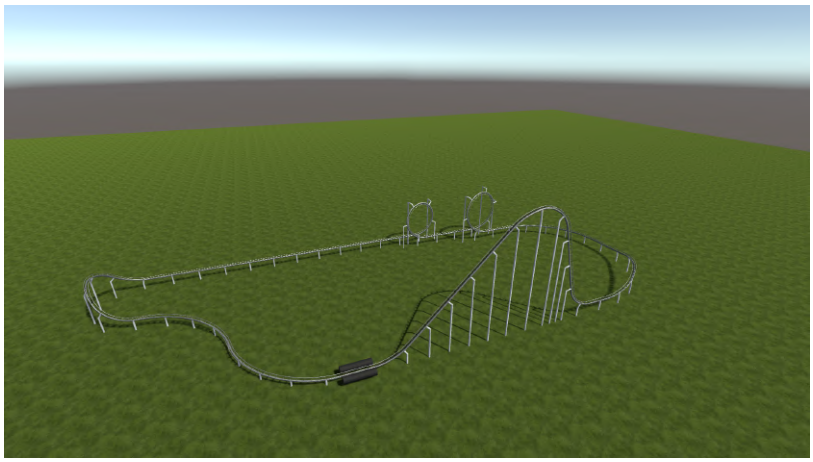


Figura 3.13: *Exemplo #2 de montanha-russa gerada no programa desenvolvido nesse projeto.*

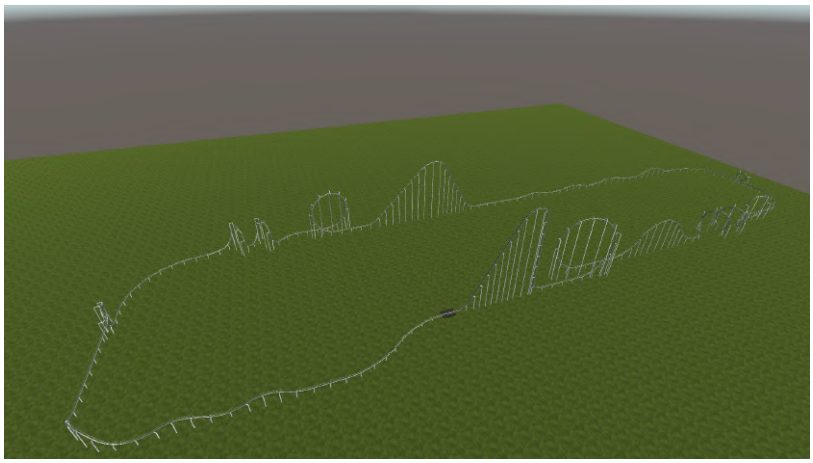


Figura 3.14: *Exemplo #3 de montanha-russa gerada no programa desenvolvido nesse projeto.*

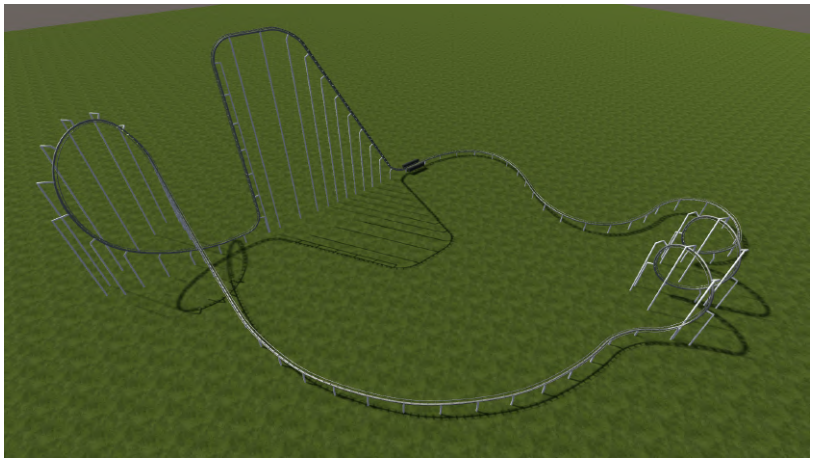


Figura 3.15: *Exemplo #4 de montanha-russa gerada no programa desenvolvido nesse projeto.*

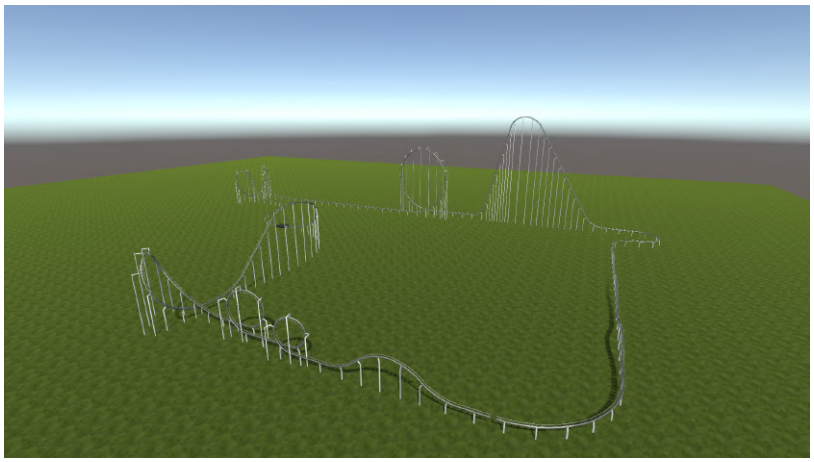


Figura 3.16: *Exemplo #5 de montanha-russa gerada no programa desenvolvido nesse projeto.*

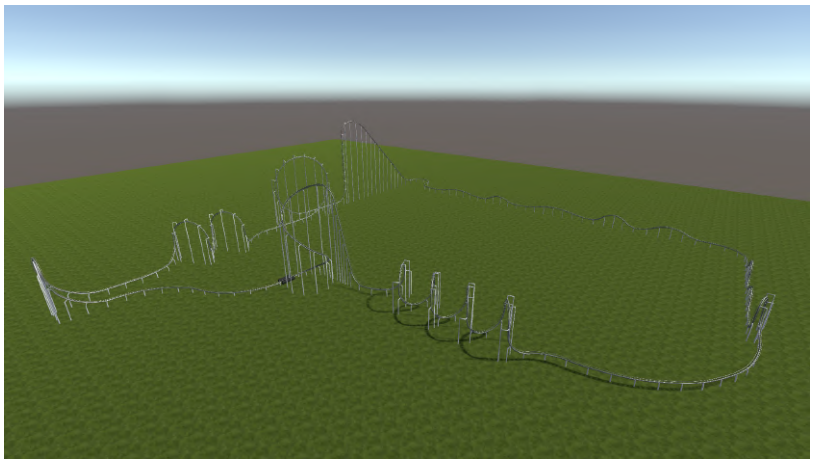


Figura 3.17: *Exemplo #6 de montanha-russa gerada no programa desenvolvido nesse projeto.*

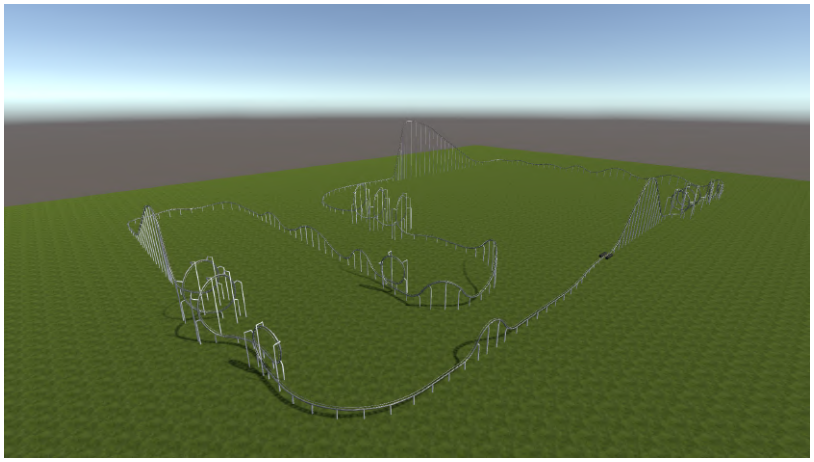


Figura 3.18: *Exemplo #7 de montanha-russa gerada no programa desenvolvido nesse projeto.*

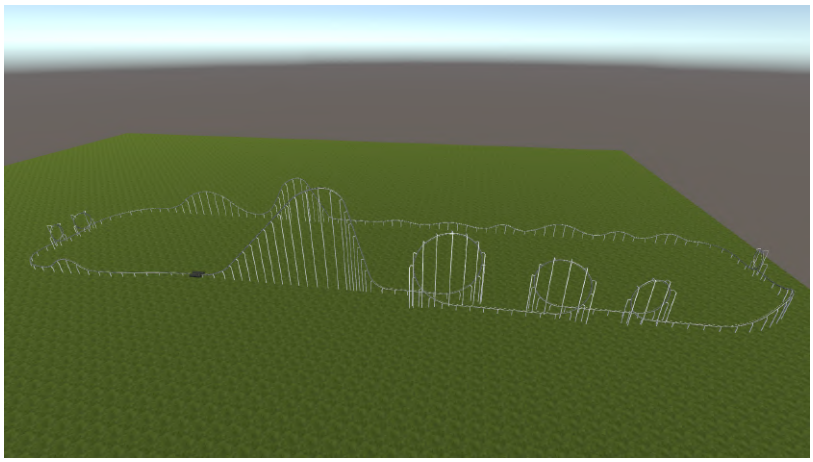


Figura 3.19: *Exemplo #8 de montanha-russa gerada no programa desenvolvido nesse projeto.*

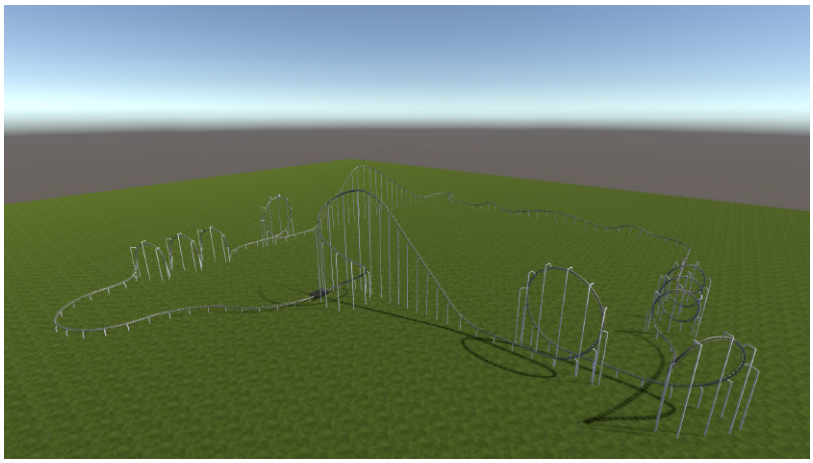


Figura 3.20: *Exemplo #9 de montanha-russa gerada no programa desenvolvido nesse projeto.*

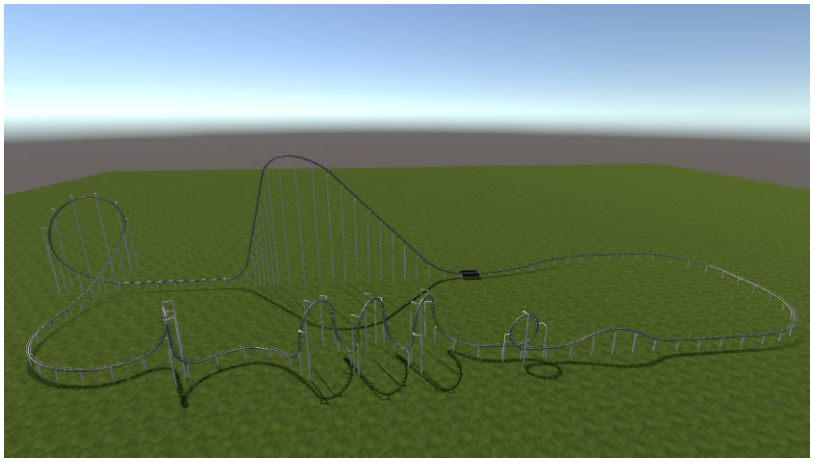


Figura 3.21: *Exemplo #10 de montanha-russa gerada no programa desenvolvido nesse projeto.*

Capítulo 4

Conclusão

4.1 Considerações finais

O RollerCoasterGenerator teve como objetivo desenvolver uma aplicação que permitisse ao usuário construir ou gerar proceduralmente sua própria montanha-russa virtual, podendo simular o carro trafegando ao longo dos trilhos. O programa também permitiria a edição do relevo do terreno e sua decoração. Portanto, esse trabalho atingiu seus objetivos propostos.

O programa desenvolvido sofreu forte influência do jogo Planet Coaster®, tendo uma qualidade comparável ao seu construtor de montanhas-russas, dado que este trabalho foi feito em seis meses por duas pessoas e o Planet Coaster® foi feito por diversas pessoas em mais tempo.

As ferramentas auxiliares à construção da montanha-russa superaram as expectativas iniciais, pois aumentaram bastante a experiência dos usuários que testaram a aplicação desenvolvida e facilitaram a construção das montanhas-russas. A parametrização dos trechos pré-prontos de trilho permitiu que usuários construíssem montanhas-russas mais realistas com grande variedade e liberdade de personalização. A ferramenta de mapa de calor da velocidade permitiu que fosse verificado se o carro completaria os trajetos dos segmentos de trilho sem que o jogador fosse obrigado a realizar a simulação completa do carro diversas vezes durante a construção.

A geração procedural de montanhas-russas atendeu a proposta desse trabalho, pois foi possível gerar montanhas-russas virtuais que respeitassem as três regras impostas inicialmente, ou seja, os trilhos não se intersectam, os trajetos das montanhas-russas são cíclicos e os carros são capazes de completá-las.

Por fim, foi conduzida uma pesquisa com possíveis usuários do RollerCoasterGenerator a fim de avaliá-lo e o software obteve um bom desempenho, sendo elogiado pela maioria dos usuários entrevistados. Vale, também, citar que este trabalho promoveu a contratação de seu autor em um emprego na área de computação gráfica e de realidade virtual.

4.2 Trabalhos futuros

Com o aumento da popularidade de óculos de realidade virtual, seria economicamente interessante adaptar a aplicação desenvolvida nesse trabalho para ter suporte à realidade virtual, permitindo que o jogador imerja muito mais no cenário de sua montanha-russa e na simulação do carro, podendo, realmente, se sentir na montanha-russa.

A fim de atrair mais jogadores, seria interessante criar novos modelos de trilhos para ter uma maior variedade de montanhas-russas. Também seria interessante que esses novos modelos fossem mais realistas. Novos modelos de carros e de objetos de cenário também aumentariam a retenção dos jogadores.

O autor desse trabalho não encontrou nenhum outro trabalho de geração procedural de montanhas-russas, considerando uma possibilidade para uma nova linha de pesquisa. A modelagem do gerador pode ser desenvolvida para se tornar mais elaborada para que as montanhas-russas geradas sejam mais próximas às reais. Também podem ser feitos mais trechos de trilhos pré-prontos para enriquecer o trajeto das montanhas-russas.

Apêndice A

Manual

A.1 Movimentação da câmera

Para movimentar a câmera, você pode utilizar as teclas “W”, “A”, “S”, “D”, ou as setas de seu teclado, ou mover o cursor para a borda da tela na direção que você quer mover ou segurar o botão direito do *mouse* e arrastá-lo na direção que você deseja mover a câmera.

Para aproximar ou afastar a câmera, basta girar o botão central do *mouse* (a “bolinha” do *mouse*).

Para rotacionar a câmera, baste segurar o botão central do *mouse* e arrastá-lo.

A.2 Menu principal

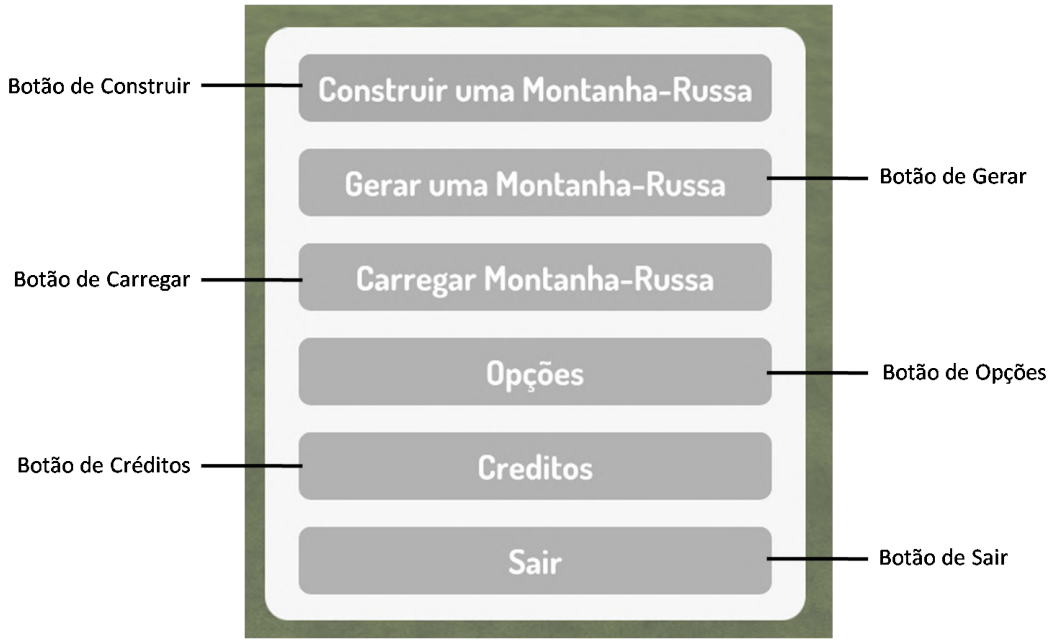


Figura A.1: *Menu Principal do jogo.*

Ao iniciar o jogo, a primeira coisa que você verá será o **Menu Principal**, mostrado na figura A.1. O **Menu Principal** contém seis botões.

O **Botão de Construir** serve para inicializar o modo de construção de sua montanha-russa, te levando para a tela da seção A.4 para você escolher o tipo da sua montanha-russa.

O **Botão de Gerar** serve para inicializar o modo de geração de sua montanhas-russas, te levando para a tela da seção A.4 para você escolher o tipo da sua montanha-russa.

O **Botão de Carregar** serve para você carregar uma montanha-russa salva anteriormente, te levando para a tela da seção A.10.

O **Botão de Opções** serve para você ajustar algumas opções do jogo, como idioma, sincronização vertical e modo de arrasto das setas de suporte, te levando para a tela da seção A.3.

O **Botão de Créditos** te mostra os créditos do jogo.

O **Botão de Sair** fecha o jogo **sem salvar** as alterações feitas por você.

A.3 Menu de opções

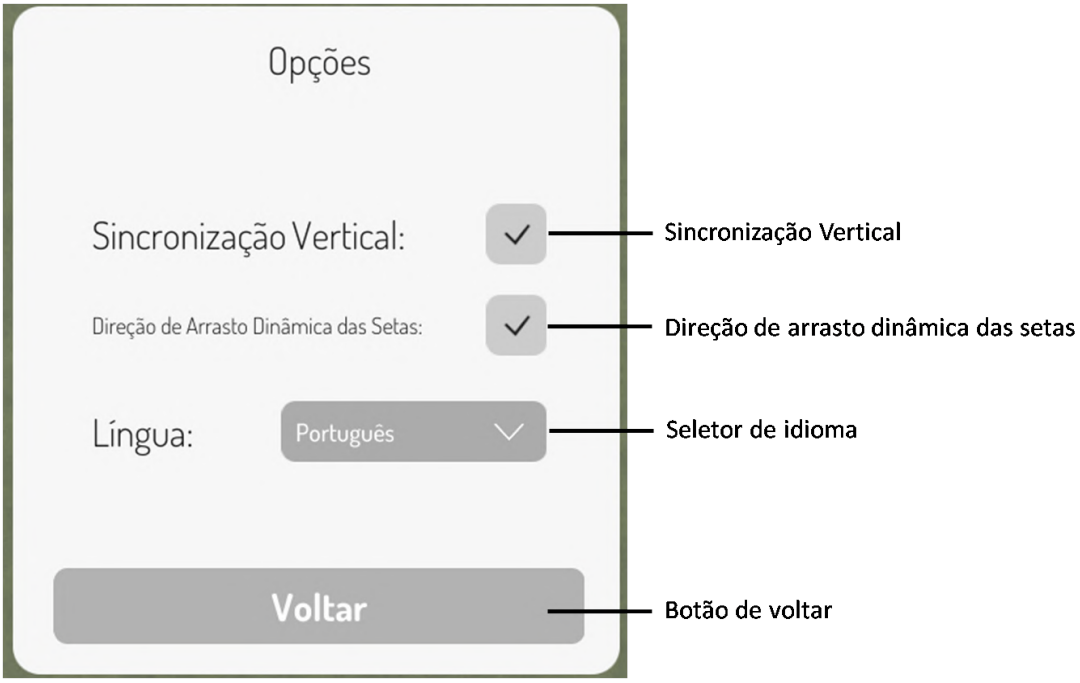


Figura A.2: *Menu de opções do jogo.*

O menu de opções, mostrado na figura A.2, pode ser acessado através do **Menu Principal**, seção A.2, e através do **Pause**, seção A.8.

A **Sincronização Vertical** sincroniza o desenho do jogo com a taxa de atualização do monitor.

A **Direção de arrasto dinâmica das setas** muda a direção que você tem que arraster as setas para alterar o formato do segmento de trilho. Se ativada, você tem que arrastar a seta em uma das direções que ela aponta a cada momento. Se desativada, você tem que arrastar apenas em uma das direções inicialmente apontadas por ela quando você clicou nela.

O **Seletor de idioma** serve para selecionar o idioma do jogo.

O **Botão de voltar** volta para o **Menu Principal**, seção A.2, caso você estivesse nele ou volta para o **Pause**, seção A.8, caso contrário.

A.4 Seletor de montanhas-russas

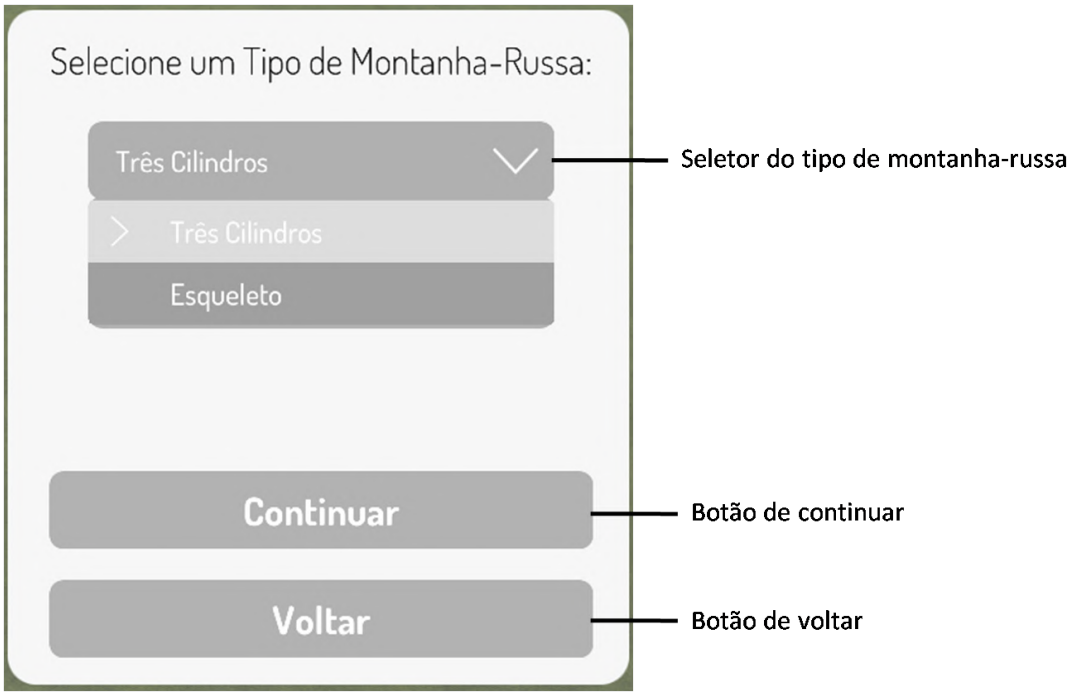


Figura A.3: *Seletor de tipos de montanhas-russas.*

O **Seletor de montanhas-russas**, mostrado na figura A.3, apresenta um seletor e dois botões e pode ser acessado através dos botões **Botão de Construir** e **Botão de Gerar** do **Menu Principal** da seção A.2.

O **Seletor do tipo de montanha-russa** serve para você selecionar o tipo de montanha-russa que quer construir.

O **Continuar** serve para construir sua montanha-russa, caso você tenha selecionado para construí-la, ou para gerá-la caso contrário.

O **Voltar** serve para voltar ao **Menu Principal**, seção A.2.

A.5 Construtor

O **Construtor de montanhas-russas** pode ser acessado através do **Seletor de montanhas-russas**, seção A.4. Ao acessá-lo, você verá algo semelhante à tela mostrada na figura A.4.

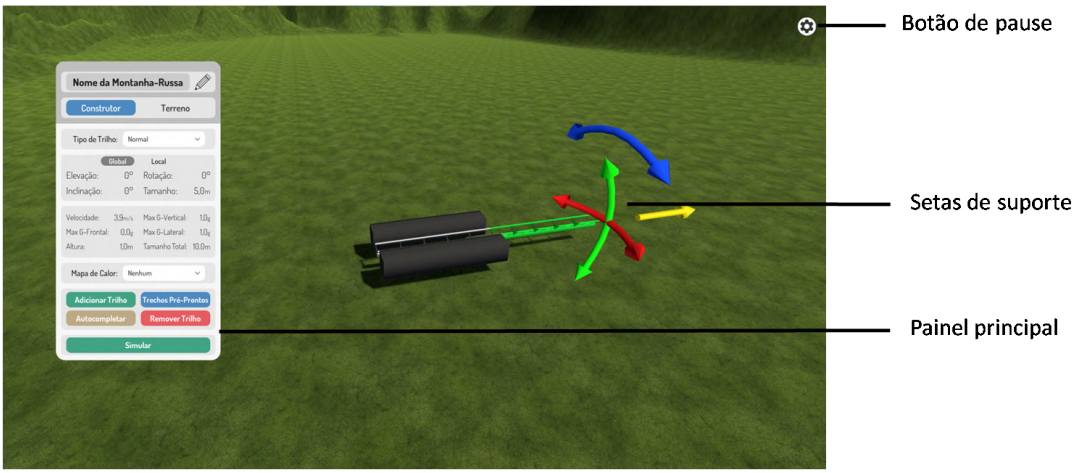


Figura A.4: *Tela de construção da montanha-russa.*

O **Botão de pause** pausa o jogo e abre o **Menu de Pause**, seção A.8.

As **Setas de suporte** servem para você alterar a forma do segmento de trilho que está sendo construído (em verde), mostradas na figura A.5. Caso o segmento de trilho que está sendo construído esteja verde, quer dizer que ele pode ser construído. Caso ele esteja vermelho, quer dizer ele não pode ser construído (provavelmente por estar intersectando com outro segmento de trilho ou com o chão).

O **Painel de construção**, mostrado na figura A.6, apresenta as principais funcionalidades para você construir sua montanha-russa. Também te dá acesso ao **Painel de terreno**, seção A.7.

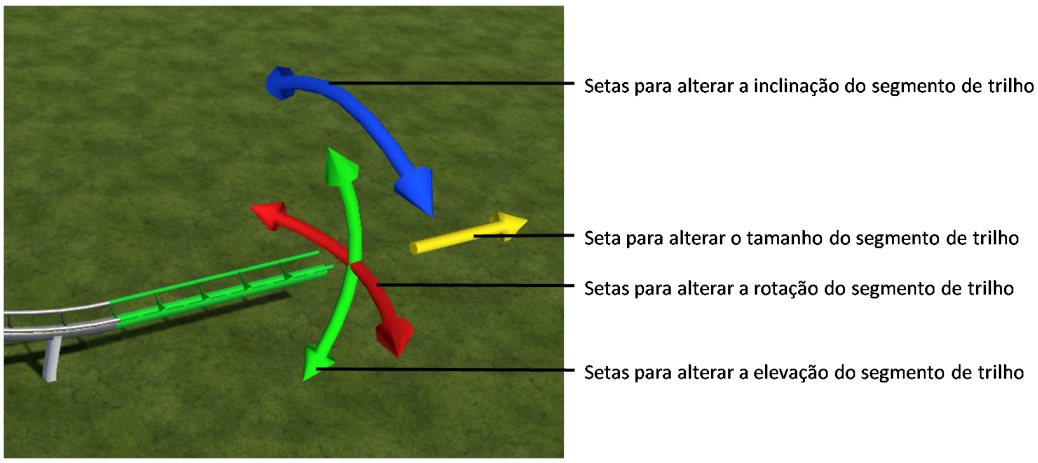


Figura A.5: *Setas de suporte à construção dos segmentos de trilho.*

As **Setas para alterar a inclinação do segmento de trilho** servem para alterar a inclinação do segmento de trilho que está sendo construído. Basta clicar nelas, arrastar o cursos na direção que você queira alterar a inclinação e soltar o cursor quando estiver satisfeito. Cada segmento pode ser inclinado no máximo 90°. Caso o segmento de trilho tenha inclinação global diferente de zero, sua elevação não poderá ser alterada.

As **Seta para alterar o comprimento do segmento de trilho** serve para alterar o comprimento do segmento de trilho que está sendo construído. Basta clicar nela, arrastar o cursos na direção que você queira alterar o comprimento e soltar o cursor quando estiver satisfeito. Os segmentos devem ter o comprimento entre 5m e 50m .

As **Setas para alterar a elevação do segmento de trilho** servem para alterar a elevação do segmento de trilho que está sendo construído. Basta clicar nelas, arrastar o cursos na direção que você queira alterar a elevação e soltar o cursor quando estiver satisfeito. Cada segmento pode ser elevado no máximo 90°.

As **Setas para alterar a rotação do segmento de trilho** servem para alterar a rotação do segmento de trilho que está sendo construído. Basta clicar nelas, arrastar o cursos na direção que você queira alterar a rotação e soltar o cursor quando estiver satisfeito. Cada segmento pode ser rotacionado no máximo 90°.

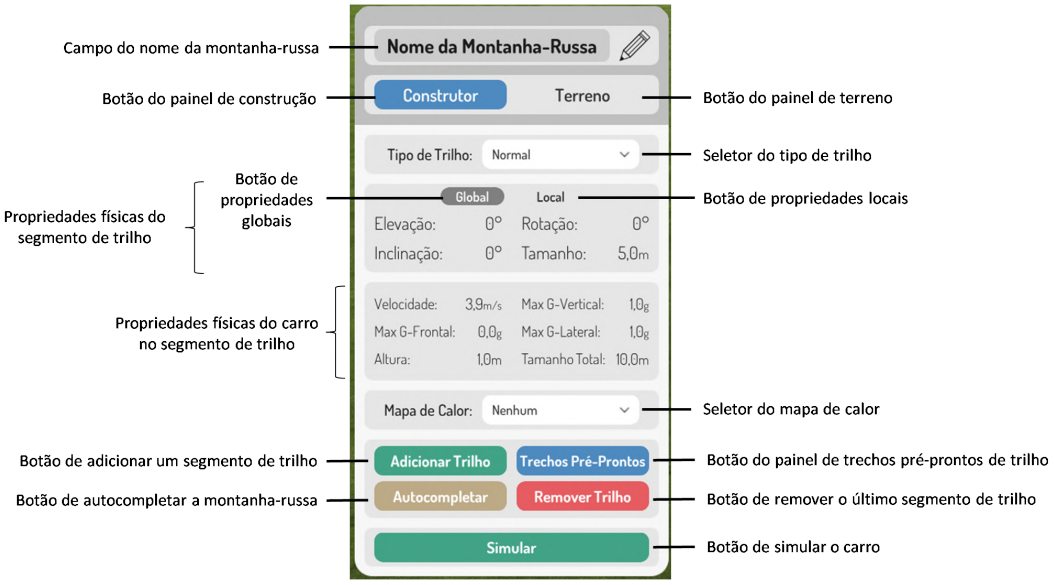


Figura A.6: *Painel de construção da montanha-russa.*

O **Campo do nome da montanha-russa** serve para alterar o nome da montanha-russa.

O **Botão do painel de construção** altera o **Painel Principal** para o **Modo de Construção**.

O **Botão do painel de terreno** altera o **Painel Principal** para o **Modo de Terreno**, seção A.7.

O **Seletor do tipo de trilho** altera o **tipo de trilho** do segmento de trilho que está sendo construído. Os tipos são: **plataforma**, **normal**, **alavanca** e **freios**.

As **Propriedades físicas do segmento de trilho** mostram as propriedades físicas, atuais, do segmento de trilho que está sendo construído. O **Botão de propriedades globais** mostra as propriedades globais e **Botão de propriedades locais** mostra as propriedades locais.

As **Propriedades físicas do carro no segmento de trilho** mostram as propriedades físicas, atuais, do carro no segmento de trilho que está sendo construído.

O **Seletor do mapa de calor** serve para mostrar os mapas de calor do carro ao longo dos trilhos.

O **Botão de adicionar um segmento de trilho** constrói o segmento de trilho que está sendo construído e adiciona um novo segmento a ser construído.

O **Botão do painel de trechos pré-prontos de trilho** abre o **Painel de trechos pré-prontos de trilho**, mostrado na figura A.7, e mostra o trecho pré-pronto de trilhos a ser construído.

O **Botão de autocompletar a montanha-russa** autocompleta a montanha-russa caso o último segmento de trilho esteja atrás da plataforma e esteja "olhando" para a plataforma.

O **Botão de remover um segmento de trilho** cancela a construição do segmento de trilho que está sendo construído, destrói o segmento anterior e adiciona um novo segmento a ser construído.

O **Botão de simular o carro** inicia a simulação do carro e muda o **Painel Principal** para o **Modo de Simulação**, seção A.6.

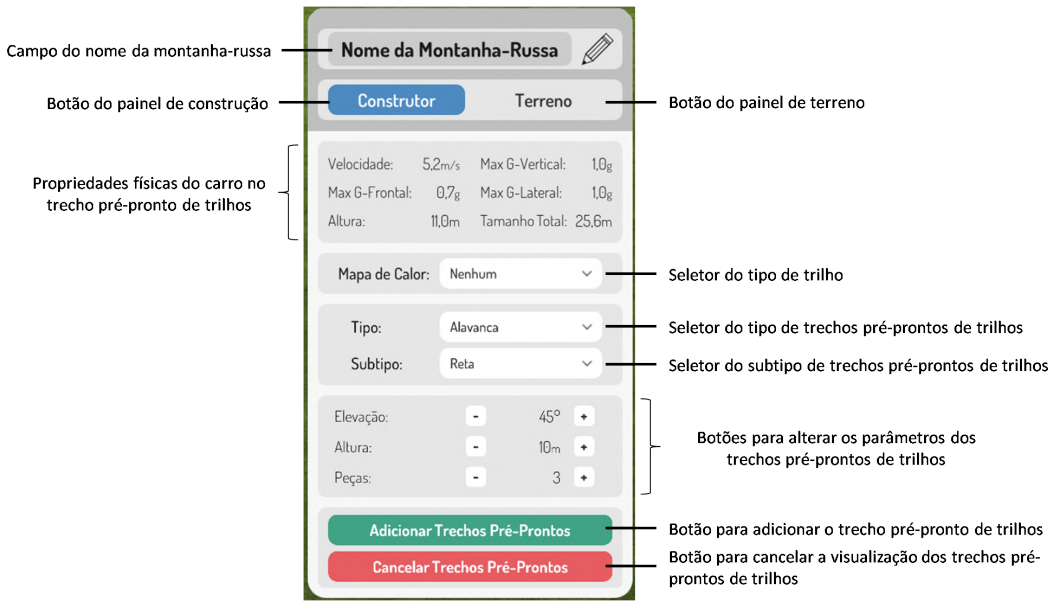


Figura A.7: *Painel de trechos pré-prontos de trilho.*

O **Seletor do tipo de trechos pré-prontos de trilhos** serve para selecionar o tipo de trecho pré-pronto de trilhos que você dejesa construir.

O **Seletor do subtipo de trechos pré-prontos de trilhos** serve para selecionar o subtipo de trecho pré-pronto de trilhos que você dejesa construir.

Os **Botões para alterar os parâmetros dos trechos pré-prontos de trilhos** servem para aumentar ou diminuir os parâmetros do trecho pré-pronto de trilhos que você está

construindo.

O **Botão para adicionar o trecho pré-pronto de trilhos** constrói o trecho pré-pronto de trilho com os parâmetros adotados.

O **Botão para cancelar a visualização dos trechos pré-prontos de trilhos** cancela a construção do trecho pré-pronto de trilhos e altera o **Painel Principal** para o **Modo de Construção**.

A.6 Simulador

Figura A.8: *Painel do simulador*

O **Painel Principal** no **Modo de Simulação**, mostrado na figura A.8, pode ser acessado através do **Painel Principal** no **Modo de Cosntrução**, seção A.5, ou no modo **Modo de Geração**, seção A.11.

O **Campo do nome da montanha-russa** serve para alterar o nome da montanha-russa.

O **Botão do painel de construção** altera o **Painel Principal** para o **Modo de Construção**.

O **Botão do painel de terreno** altera o **Painel Principal** para o **Modo de Terreno**, seção A.7.

As **Propriedades físicas do carro** mostram as propriedades físicas, atuais, do carro que está sendo simulado.

O **Seletor do mapa de calor** serve para mostrar os mapas de calor do carro ao longo dos trilhos.

O **Botão de mudar a câmera** altera o modo da câmera para **passageiro do carro** ou para **voando**.

O **Botão de para a simulação do carro** para a simulação do carro e retora o **Painel Principal** para o modo anterior: **Modo de Construção**, seção A.5, ou **Modo de Geração**, seção A.11.

A.7 Terreno

O **Painel Principal** no **Modo de Terreno** pode ser acessado através do **Painel Principal** nos modos **Modo de Cosntrução**, seção A.5, **Modo de Geração**, seção A.11 e **Modo de Simulação**, seção A.6.

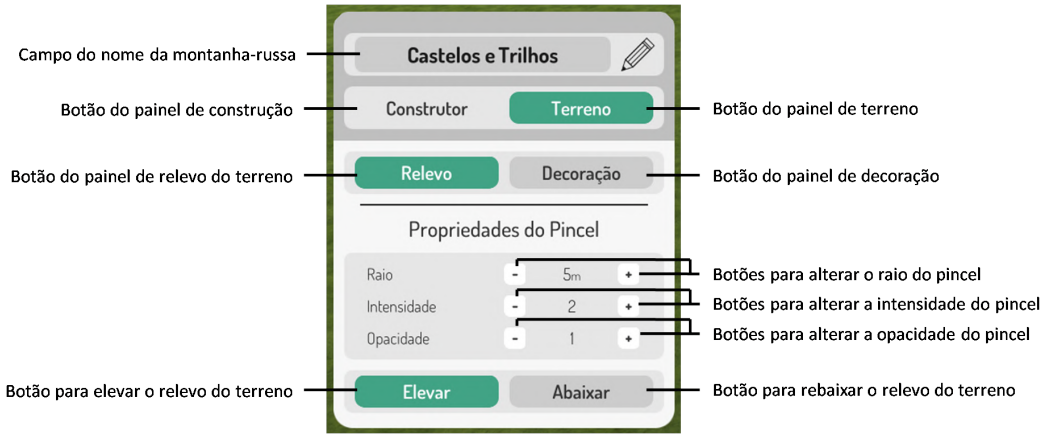


Figura A.9: *Painel do relevo do terreno*

O **Painel Principal** no **Modo de Relevo**, mostrado na figura A.9 é acessado apertando o **Botão do painel de relevo do terreno**.

O **Campo do nome da montanha-russa** serve para alterar o nome da montanha-russa.

O **Botão do painel de construção** altera o **Painel Principal** para o **Modo de Construção**.

O **Botão do painel de terreno** altera o **Painel Principal** para o **Modo de Terreno**, seção A.7.

O **Botão do painel de relevo do terreno** altera o **Painel Principal** para o **Modo de Relevo**.

O **Botão do painel de decoração** altera o **Painel Principal** para o **Modo de Decoração**.

Para alterar o relevo do terreno, no **Modo de Relevo**, basta clicar onde você quer alterar o relevo no terreno que o **Pincel do Relevo** irá alterar o relevo.

Os **Botões para alterar o raio do pincel** altera o raio do **Pincel do Relevo**.

Os **Botões para alterar a intencidade do pincel** altera a intencidade do **Pincel do Relevo**.

Os **Botões para alterar a opacidade do pincel** altera a opacidade do **Pincel do Relevo**.

O **Botão para elevar o relevo do terreno** faz com que o **Pincel do Relevo** eleve o relevo do terreno.

O **Botão para rebaixar o relevo do terreno** faz com que o **Pincel do Relevo** rebaixe o relevo do terreno.

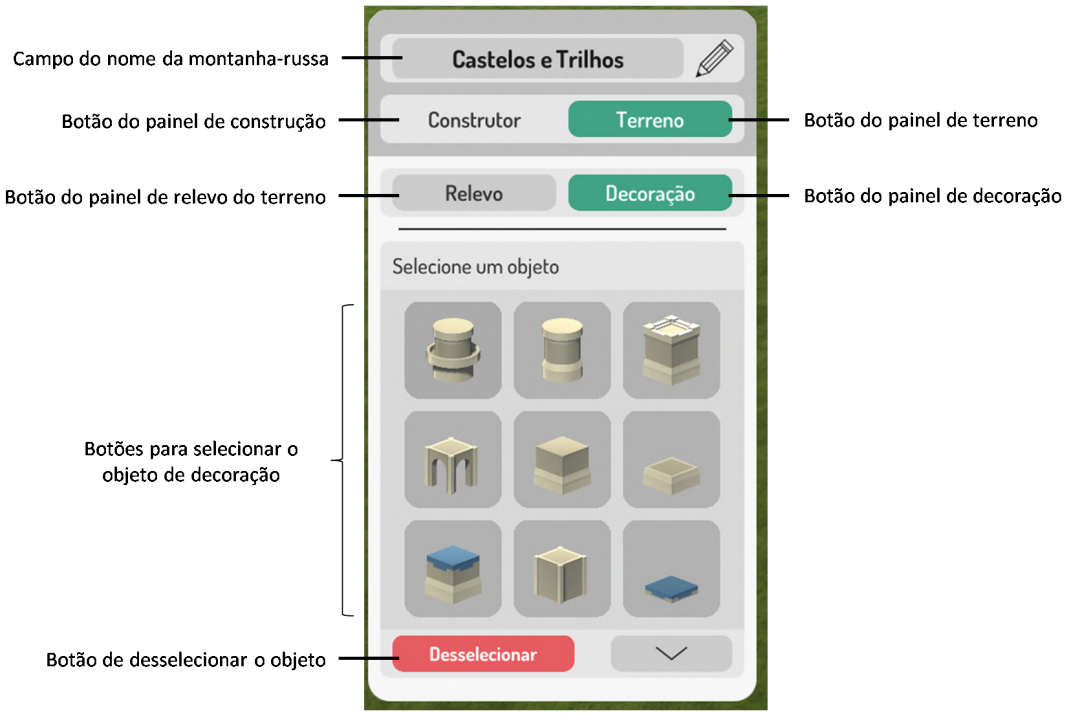


Figura A.10: *Painel dos objetos de decoração*

O **Painel Principal** no **Modo de Decoraçã**, mostrado na figura **??** é acessado apertando o **Botão do painel de decoração**.

Os **Botões para selecionar o objeto de decoração** servem para você selecionar qual objeto de decoração você quer colocar no terreno.

O **Botão de desselecionar o objeto** serve para você desselecionar o objeto que está atualmente selecionado.

Para colocar um objeto, basta selecionar qual você quer colocar, posicionar o cursor para onde você quer colocá-lo e clicar. Para rotacioná-lo, aperte a tecla “R” Para elevá-lo, aperte a tecla “E” e, para rebaixá-lo, aperte a tecla “Q”.

Para deletar um objeto, basta selecioná-lo e apertar o **Botão de desselecionar o objeto**.

O **Botão de página anterior** muda a página dos objetos para a anterior e o **Botão de próxima página** muda para a próxima.

A.8 Pause

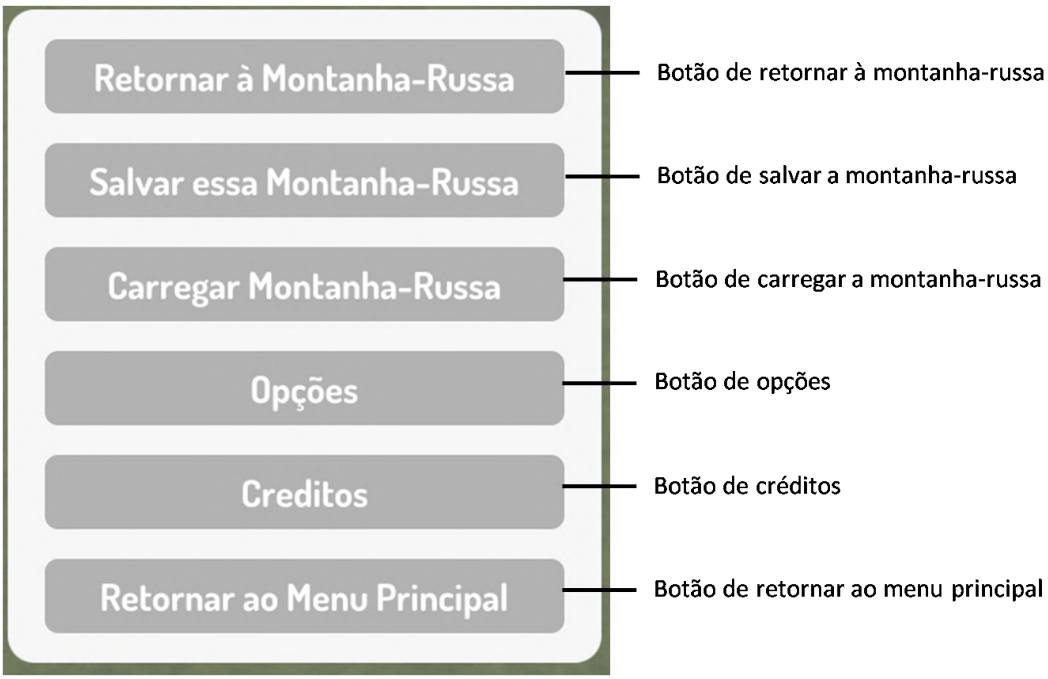


Figura A.11: *Menu de pause.*

O **Menu de Pause**, mostrado na figura A.11, pode ser acessado apertando a tecla “Esc” ou o **Botão de pause** da tela. Ele apresenta seis botões.

O **Botão retornar à montanha-russa** tira o pause do jogo e fecha o **Menu de Pause**.

O **Botão de salvar a montanha-russa** serve para salvar a montanha-russa, te levando para a tela da seção A.9.

O **Botão de Carregar** serve para você carregar uma montanha-russa salva anteriormente, te levando para a tela da seção A.10.

O **Botão de Opções** serve para você ajustar algumas opções do jogo, como idioma, sincronização vertical e modo de arrasto das setas de suporte, te levando para a tela da seção A.3.

O **Botão de Créditos** te mostra os créditos do jogo.

O **Botão de retornar ao menu principal** retorna ao **Menu Principal**, seção A.1, sem salvar as alterações feitas.

A.9 Salvar

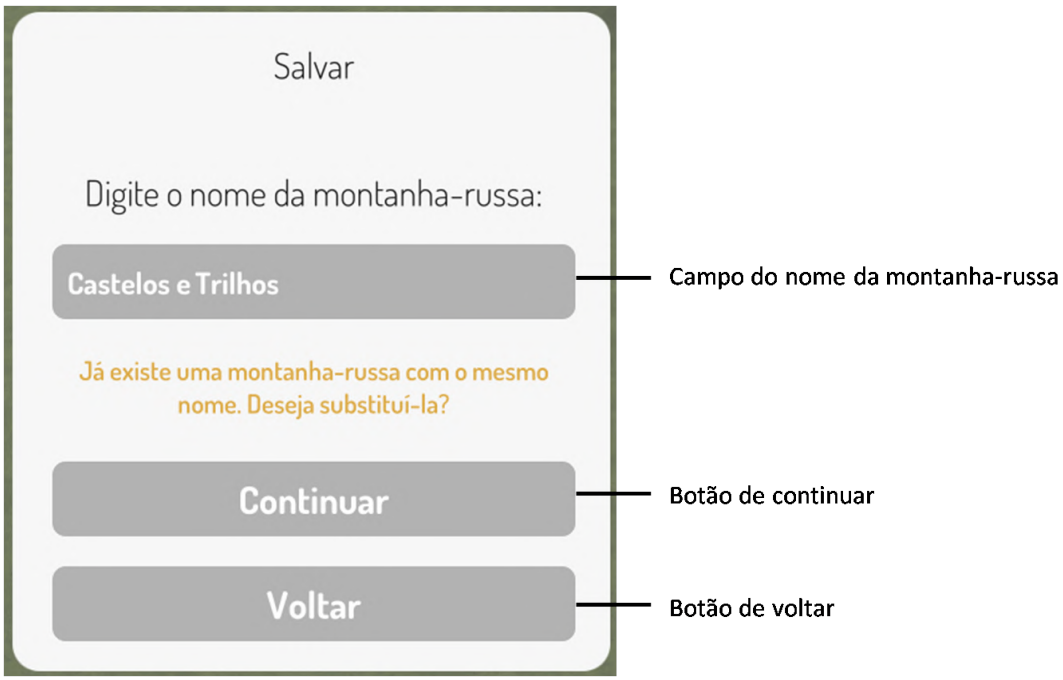


Figura A.12: *Painel de salvar montanhas-russas.*

O **Painel de Salvar**, como mostrado na figura A.13, pode ser acessado pelo **Menu de Pause**, seção A.11.

O **Campo do nome da montanha-russa** serve para alterar o nome da montanha-russa.

O **Botão de continuar** lhe fará tirar uma foto, como na figura A.13, para salvar sua montanha-russa.

O **Botão de voltar** cancela o salvamento da montanha-russa e volta para o **Menu de Pause**, seção A.11.

Figura A.13: *Câmera de foto para salvar a montanha-russa.*

O **Botão de tirar a foto** tira a foto e salva a montanha-russa junto com o relevo do terreno e com os objetos.

A.10 Carregar

Figura A.14: *Painel de carregar montanhas-russas.*

O **Menu de Carregar**, mostrado na figura A.14, pode ser acessado através do **Menu Principal**, seção A.1, e do **Menu de Pause**, seção A.11.

O **Seletor de montanhas-russas salvas** serve para selecionar qual montanha-russa salva você quer carregar.

O **Botão de carregar** carrega a montanha-russa selecionada.

O **Botão de voltar** retorna ao **Menu Principal**, seção A.1, ou ao **Menu de Pause**, seção A.11.

A.11 Gerador

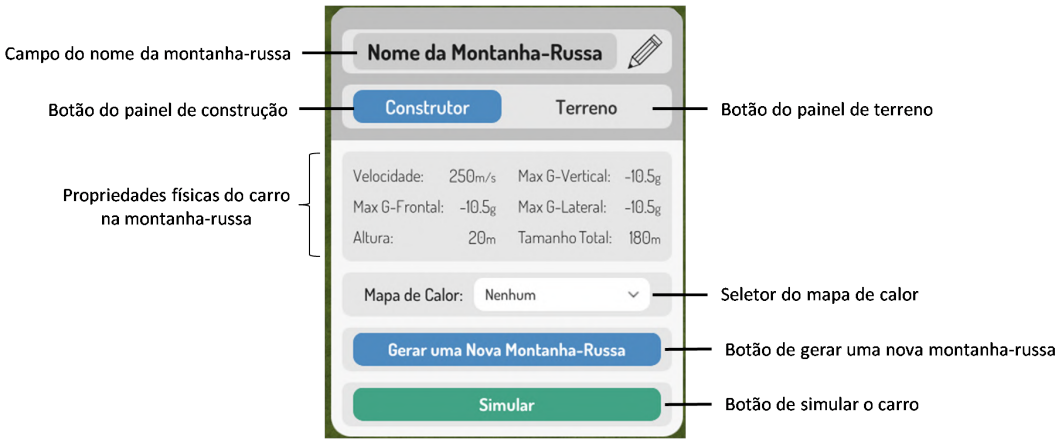


Figura A.15: *Painel Principal no Modo Gerador.*

O **Painel Principal** no **Modo Gerador** pode ser acessado através do **Botão de Gerar** do **Menu Principal**, seção A.2.

O **Campo do nome da montanha-russa** serve para alterar o nome da montanha-russa.

O **Botão do painel de construção** altera o **Painel Principal** para o **Modo de Construção**.

O **Botão do painel de terreno** altera o **Painel Principal** para o **Modo de Terreno**, seção A.7.

As **Propriedades físicas do carro na montanha-russa** mostram as propriedades físicas do carro na montanha-russa gerada.

O **Seletor do mapa de calor** serve para mostrar os mapas de calor do carro ao longo dos trilhos.

O **Botão de gerar uma nova montanha-russa** gera uma nova montanha-russa.

O **Botão de simular o carro** inicia a simulação do carro e muda o **Painel Principal** para o **Modo de Simulação**, seção A.6.

Referências

[Acks *et al.* 2020] William Acks *et al. This is War*. Mai. de 2020 (citado na pg. 1).

[Hanan 1992] James Scott Hanan. *Parametric L-systems and their application to the modelling and visualization of plants*. 1992 (citado nas pgs. 4, 35).

[Harris e Threewitt 2007] Tom Harris e Cherise Threewitt. *How Roller Coasters Work*. https://science.howstuffworks.com/engineering/structural/roller-coaster.htm. Último acesso em 05/07/2021. 2007 (citado na pg. 1).

[Hocevar 2018] Sam Hocevar. *How to achieve uniform speed of movement on a bezier curve?* https://gamedev.stackexchange.com/questions/27056/how-to-achieve-uniform-speed-of-movement-on-a-bezier-curve. Último acesso em 05/07/2021. 2018 (citado na pg. 15).

[Hyková 2010] Tereza Hyková. "Roller Coaster Simulator". English. Master's thesis. Czech Technilcal University in Prague. Faculty of Electricel Engineering, 2010, pg. 62. URL: https://dcgi.fel.cvut.cz/theses/2010/hykovter (citado nas pgs. 3, 6).

[Lengyel 2011] Eric Lengyel. *Mathematics for 3D Game Programming and Computer Graphics, Third Edition*. 3rd. Boston, MA, USA: Course Technology Press, 2011. ISBN: 1435458869 (citado nas pgs. 6, 20).

[*List of roller coaster elements* s.d.] *List of roller coaster elements*. https://coasterpedia.net/wiki/List_of_roller_coaster_elements. Último acesso em 05/07/2021 (citado na pg. 28).

[Marschner e Shirley 2016] Steve Marschner e Peter Shirley. *Fundamentals of Computer Graphics, Fourth Edition*. 4th. USA: A. K. Peters, Ltd., 2016. ISBN: 1482229390 (citado na pg. 15).

[Park 2003] Sang C. Park. "Polygonal extrusion". Em: *The Visual Computer* 19.1 (mar. de 2003), pgs. 38–49. ISSN: 1432-2315. DOI: 10.1007/s00371-002-0170-2. URL: https://doi.org/10.1007/s00371-002-0170-2 (citado na pg. 11).

[Riškus 2006] Aleksas Riškus. "Approximation of a cubic bezier curve by circular arcs and vice versa". Em: *Information Technology and Control* 35 (jan. de 2006) (citado na pg. 8).

[Väisänen 2018] Antti Väisänen. “Design of Roller Coasters”. English. Master’s thesis. Aalto University. School of Engineering, 2018, pg. 75. URL: http://urn.fi/URN:NBN:fi:aalto-201809034831 (citado na pg. 5).