Desenvolvimento de um jogo educacional para o ensino da geometria fractal

Gabriel Neis Leão[1], Gustavo Stangherlin Cantarelli[1]

[1]Sistemas de Informação – Centro Universitário Franciscano
Santa Maria – RS – Brasil

g.leao@unifra.edu.br, gus.cant@gmail.com

Abstract. *The computer has become an indispensable tool at the time of the studies. With that in mind, this work shows the development of an educational game for the basic education of fractal geometry, using as key differentiators 3D graphics, animations and a soundtrack. The development resulted in a software that surpassed expectations, not only covering the fractal geometry, but also the flat geometry.*

Resumo. *O computador tornou-se uma ferramenta indispensável na hora dos estudos. Considerando isso, este trabalho mostra o desenvolvimento de um jogo educativo para o ensino básico da geometria fractal, utilizando gráficos tridimensionais, animações e uma trilha sonora como principais diferenciais. O desenvolvimento resultou em um software que superou as expectativas, não só abrangendo a geometria fractal, mas também a geometria plana.*

1. Introdução

A história da ciência da matemática é de longa data e de grande importância para a humanidade, pois, a partir dos cálculos, os engenheiros podem construir prédios seguros, carros de alta *performance*, equipamentos para o auxílio da saúde etc. Blaise Pascal construiu a primeira máquina de calcular em 1642 [Pacievitch 2016], e os poderosos computadores da atualidade nasceram da ideia de fazer uma máquina de calcular. Com o crescimento da tecnologia, a popularização dos computadores e a facilidade de acesso à Internet, uma certa porcentagem da população está tornando-se mais acomodada, afinal, hoje em dia, os computadores fazem quase tudo para uma pessoa. Assim, o ensino em escolas, principalmente no ensino fundamental, está começando a sentir o desinteresse das crianças em aprender o básico e necessário.

Em geral, as crianças tendem a não gostar da matemática ou ter dificuldades em aprendê-la. Para mudar isso, deve-se ter uma nova concepção sobre a matemática, de que ela não é difícil, mas sim divertida, lógica e está em todo lugar. Vários objetos podem ser feitas por meio de cálculos, e uma maneira de demonstrar isso é analisando tudo o que há ao nosso redor. Em qualquer lugar, a matemática é encontrada, implícita ou explicitamente. A geometria fractal trata exatamente disso. Essa geometria mostra que a matemática existe em forma de padrões, desde a natureza até o caos [Barbosa 2002].

Por meio da geometria fractal, podem ser calculadas áreas de terrenos não planos e observadas autossemelhanças entre rochas. Além disso, ela também pode ser aplicada

na arte (em pinturas e até na música). Conhecendo a importância da geometria fractal, foi elaborada uma proposta que poderia facilitar o estudo, o ensino e o entendimento por meio do computador, o qual, nos tempos modernos, vem auxiliando o ensino de forma geral.

Este trabalho conta com a colaboração do curso de Matemática, do Centro Universitário Franciscano, a qual é de importância para todo o desenvolvimento da ferramenta e para o desenvolvimento do referencial teórico pela perspectiva da matemática.

1.1. Objetivos gerais

Este trabalho tem como objetivo desenvolver um jogo eletrônico na plataforma Web para o ensino da geometria fractal. O jogo será uma ferramenta para auxiliar no aprendizado, buscando transformar o que é potencialmente complexo em algo simples e divertido.

1.2. Objetivos específicos

Os objetivos específicos deste trabalho são:

- Realizar estudos sobre geometria fractal;
- Realizar estudos sobre desenvolvimento de jogos;
- Realizar estudos sobre jogos na educação;
- Utilizar a criatividade para o desenvolvimento de um software (jogo eletrônico) para o auxílio no ensino da geometria fractal;
- Aprender a utilização de interfaces de desenvolvimento gráfico, *engines* e desenvolvimento de jogos eletrônicos;

1.3. Estrutura do trabalho

O trabalho está organizado da seguinte forma: na Seção 2, é apresentada uma revisão bibliográfica, demonstrando as pesquisas e a fundamentação teórica que levaram à escolha das ferramentas de desenvolvimento; na Seção 3, referenciam-se alguns trabalhos relacionados que auxiliaram no incentivo e na concepção da solução sugerida; na Seção 4, a solução proposta é apresentada de forma detalhada, passando pelas fases de desenvolvimento; na Seção 5, é apresentado o projeto, mostrando o software desenvolvido; e, na Seção 6, são apresentadas as conclusões e considerações finais.

2. Revisão bibliográfica

Esta seção apresenta as pesquisas que foram realizadas, baseando-se em livros, trabalhos correlatos e artigos.

2.1. Jogos digitais na educação

A ideia de trazer jogos de computador para a educação não é novidade. Na década de 1990, uma desenvolvedora de *softwares* educacionais, chamada Davidson & Associates, lançou, no mercado, o jogo Math Blaster [Davidson & Associates 1993]. Neste, o jogador é um astronauta que tem como missão salvar seu amigo, e, para avançar no

jogo, ele precisa fazer cálculos simples. Math Blaster foi um marco histórico para os jogos educacionais, pois foi um dos pioneiros a alcançar recordes de vendas em todo o mundo. Existem outros jogos que também quebraram paradigmas, como o Coelho Sabido [The Learning Company 1999], um jogo infantil que tem como público-alvo crianças da pré-escola e dos anos iniciais do ensino fundamental. Neste, o jogador pode explorar os cenários e deve alcançar alguns objetivos. Para isso, o jogo impõe pequenos desafios, como escolher uma figura solicitada ou resolver um jogo da memória.

2.1.1. *Gamification*

Consiste em aplicar técnicas de *design* utilizadas em jogos em um contexto que não possui características de jogo. Pela sua aplicação, é possível transformar o que seria uma tarefa repetitiva e cansativa em uma experiência completamente diferente e divertida, acabando com a desmotivação [Alves 2015].

Os principais elementos do *Gamification* são: fantasia, objetivos claros, retorno e ajuda, liberdade de acordo com o nível de aprendizado, pressão causada por um tempo limite e recompensas por ter atingido as etapas. Segundo Alves (2015), *Gamification* não é a transformação de qualquer atividade em um jogo, mas sim aprender a partir deles, encontrar elementos que possam melhorar uma experiência sem desprezar o mundo real e reconhecer o poder da diversão.

2.2. Desenvolvimento de jogos digitais

Os jogos eletrônicos, desde o seu surgimento, trazem entretenimento a todos os que os veem ou jogam, tanto com a diversão que proporcionam quanto com o realismo gráfico. Para tanto, as áreas mais visíveis são a parte de enredo e imersão, criada pelos artistas, e a parte de jogabilidade, desenvolvida e melhorada pelos programadores e desenvolvedores.

Segundo Perucia et al. (2007), para que um jogo seja desenvolvido, são necessários vários tipos de profissionais, sendo, pelo menos, um de cada área. São eles: programadores (encarregados de desenvolver o *software*); artistas (responsáveis pela parte de *layout* do jogo, criam os desenhos dos personagens, ilustrações etc.); projetistas de níveis/fases (elaboram os desafios de cada nível/fase); projetistas de jogos (mais conhecidos como *Game Designers*, têm uma visão geral sobre todo o projeto, geram o *design document*); planejador de *software* (estima o tempo e o esforço das implementações e divide as tarefas para a equipe); arquiteto-chefe (responsável pela arquitetura geral do projeto); gerente de projeto (responsável pelo cronograma do projeto e pelo monitoramento da equipe de desenvolvimento); músicos e sonoplastas (responsáveis pelas trilhas sonoras, pelas vozes e pelos efeitos sonoros); e testadores (responsáveis pela busca de falhas, travamentos e erros – *bugs*).

Após a definição dos integrantes da equipe, dá-se início ao ciclo de desenvolvimento. Neste, o primeiro passo a ser realizado é definir o que será desenvolvido. Para isso, a equipe reúne-se e começa a ter ideias e a criar conceitos por meio da técnica conhecida como *brainstorm* (uma chuva de ideias). Os integrantes devem utilizar questões como “Qual o objetivo do jogo?”, “O que o jogador terá de fazer?”, “Como ele vai fazer?”, “O que tornará o jogo divertido?”, “Qual será o público-

alvo?" etc., a fim de gerar ideias mais claras e consistentes desde o início, evitando-se, assim, alguma mudança radical no escopo do projeto [Perucia et al. 2007].

2.3. Motores de jogos para Web

Para que os gráficos sejam executados de forma consistente, o ambiente deve ser favorável e ter os recursos necessários. No caso de um jogo desenvolvido para ser executado em um navegador de internet, é necessário o uso de uma API. Para o desenvolvimento deste projeto, a API para gráficos 3D escolhida foi a WebGL.

O WebGL é um API de baixo nível que funciona em múltiplas plataformas (*cross-platform*), o qual tem base no OpenGL 2.0 e utiliza o elemento gráfico Canvas, que é nativo do HTML5, para a sua execução. A programação para o WebGL pode ser feita pela linguagem JavaScript. A grande vantagem do seu uso é a não necessidade de instalar qualquer extensão (*Plugin*) para ser executado [Khronos Group 2016].

A partir da escolha do WebGL, foram pesquisados os motores gráficos que utilizam esse API. Dentre os motores pesquisados, o que mais se destacou foi o Unity 5, pois apresenta grande quantidade de recursos e interface de desenvolvimento bem organizada, tornando ágil o desenvolvimento.

2.3.1. Unity 5

O motor gráfico Unity 5 é utilizado em várias plataformas, tanto em jogos quanto em demonstrações visuais, apresentando uma boa qualidade de imagem e renderização. Com a evolução desse motor gráfico, os desenvolvedores adicionaram suporte para o WebGL, que pode ser executado por meio de um navegador de Internet, inovando e trazendo mais recursos para os *websites* [Unity 2016].

Sua interface de desenvolvimento utiliza as linguagens de programação JavaScript e C# para o desenvolvimento. Possui um ambiente amigável de trabalho, organizado e com inúmeros recursos. Possui também uma grande comunidade de desenvolvedores independentes que disponibilizam vários exemplos, podendo ser baixados e testados pela interface de desenvolvimento, facilitando o aprendizado sobre a ferramenta. Sua aquisição é gratuita e pode ser realizada na própria página da Unity.

2.4. Geometria fractal

Na Seção 1 deste trabalho, pode-se notar o quanto foi citada a geometria em geral. A geometria fractal é a geometria que explora conceitos de infinito, detecção de padrões no meio do "caos" e visualização de autossemelhanças entre objetos da natureza. O exemplo mais simples que se pode citar de objeto autossemelhante da natureza é uma árvore. A árvore possui ramificações, as quais possuem ramificações, e estas possuem mais ramificações (os galhos). Ao analisar um galho de uma árvore, observa-se, evidentemente, que ele se parece muito com uma miniatura da árvore toda. Até nas próprias folhas, nota-se um desenho semelhante a uma árvore com suas ramificações [Barbosa 2002].

Segundo Kawashima (1998), fractais podem ser determinísticos ou não determinísticos. Um fractal determinístico é aquele cuja aparência geral é estritamente idêntica a cada vez que é gerado. Isso é garantido pelo caráter determinístico das

fórmulas matemáticas que os geram. Os fractais determinísticos também se podem chamar Sistemas de Funções Iteradas (IFS). Um fractal não determinístico é aquele que introduz elementos de natureza estocástica nas suas fórmulas matemáticas geratrizes. Ou seja, é praticamente impossível que um fractal desse tipo torne-se exatamente idêntico depois de gerado duas vezes, usando-se o mesmo processo, pois a tendência é sempre a criação de um fractal diferente. Os fractais não determinísticos também podem ser chamados de Fractais de Tempo de Escape (FTE).

Para descrever os fractais de forma mais estruturada, serão apresentados, a seguir, breve e objetivamente, dois dos mais notáveis exemplos de fractais IFS históricos e, em seguida, o Jogo do Caos.

2.4.1. Curva de Koch

A Curva de Koch é um exemplo prático, simples e muito famoso, conhecida também como a única curva sem tangente. A curva é construída a partir de uma reta. O primeiro passo é dividir a reta inicial em três retas de tamanho igual. Após isso, a partir do segmento do meio, constrói-se um triângulo equilátero e é removida a base do triângulo. Ao final da iteração, a reta encontrar-se-á transformada em quatro retas de tamanho igual, com o centro disposto em forma de triângulo (Figura 1, Nível 1). A cada nova iteração, são criadas novas ramificações, adicionando mais detalhes e apresentando uma figura com forma curiosa [Barbosa 2002]. Na Figura 1, é apresentado o exemplo descrito anteriormente.

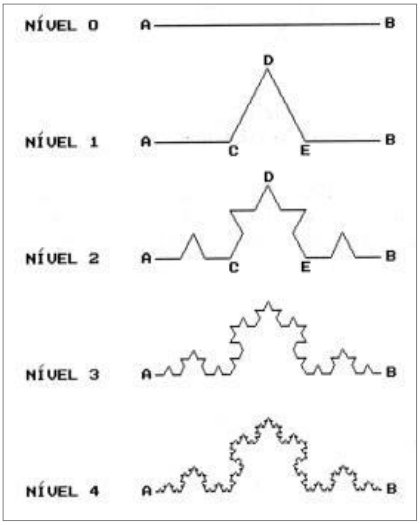


Figura 1. Exemplo do processo de construção da Curva de Koch [Lopes e Pantaleão 2016]

2.4.2. Triângulo de Sierpinski

O Triângulo de Sierpinski (Figura 2) é um fractal peculiar que pode ser reproduzido tanto no plano cartesiano quanto no plano tridimensional. A construção dessa figura é iniciada a partir de um triângulo equilátero. Em seguida, são marcados os pontos médios das arestas e é removido o triângulo central, restando três triângulos. Na próxima iteração, os três triângulos passarão pelo mesmo processo, gerando nove triângulos [Barbosa 2002].

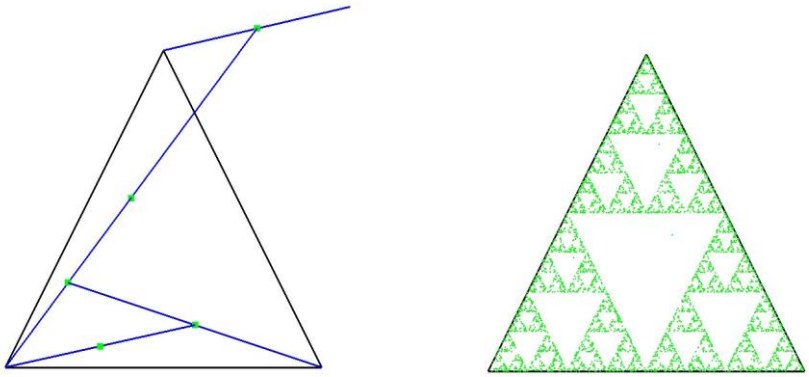
Figura 2. Exemplo das iterações do Triângulo de Sierpinski [Wikipédia 2016]

2.4.3. Jogo do Caos

O Jogo do Caos é um jogo que consiste em provar que é possível gerar um fractal a partir de dados aleatórios. Os requisitos para jogar são um triângulo equilátero desenhado no plano cartesiano, um ponto aleatório marcado no plano e um dado (de jogo de tabuleiro). O jogador seleciona o ponto aleatório e, então, joga o dado. O sorteio do dado representará o vértice a ser ligado com o ponto. Os vértices do triângulo correspondem às letras A, B e C. Se o dado cair na posição 1 ou 2, deverá ser usado o ponto A. Se o dado cair na posição 3 ou 4, deverá ser usado o ponto B. Se o dado cair na posição 5 ou 6, deverá ser usado o ponto C [Janos 2008].

Então, o jogador liga o ponto aleatório ao ponto do vértice sorteado pelo dado, gerando uma reta. Ele calcula o ponto médio dessa reta e marca-o no plano; então, sorteia novamente o dado e repete o processo a partir do novo ponto. São marcados somente os pontos, e não as retas. A partir de, aproximadamente, 75 jogadas, os pontos começam a organizar-se, e uma formação semelhante ao Triângulo de Sierpinski torna-se mais evidente [Janos 2008].

A Figura 3 (a) mostra a implementação do Jogo do Caos em OpenGL, desenvolvida pelo autor deste trabalho. No aplicativo, o jogador seleciona um número de iterações a serem executadas de forma automática, tendo a opção de observar a construção passo a passo, a partir de um ponto clicado na tela. Observa-se que, nesse exemplo, as linhas também são marcadas no plano, porém a aplicação permite que sejam ocultadas as linhas e que seja diminuído o tamanho dos pontos, aumentando a precisão da imagem. Na Figura 3 (b), é aplicada a teoria do Jogo do Caos, demonstrando o resultado da aplicação de 5.000 iterações. O programa desenvolvido para testes também dispõe de iterações automatizadas e mostra as coordenadas do ponto marcado a cada iteração e o número de iterações executadas.



(a) (b)

Figura 3. Jogo do Caos desenvolvido em OpenGL: (a) aplicando cinco iterações e mostrando as retas; (b) aplicando 5.000 iterações e não mostrando as retas

2.5. *Feature Driven Development* (FDD)

O *Feature Driven Development* foi concebido, originalmente, por Peter Coad e seus colegas como um modelo de processo prático para engenharia de *software* orientada a objeto. Como as outras metodologias ágeis, o FDD dá ênfase na colaboração entre os integrantes de uma equipe de desenvolvedores, administra a complexidade de projeto e de problemas utilizando decomposição baseada em funcionalidades e comunicação sobre detalhes técnicos, verbalmente, graficamente e por meio de texto [Pressman 2010].

Os processos "Desenvolver modelo inicial", "Criar lista de funcionalidades" e "Planejar por funcionalidade" têm a visão geral e inicial de todo o projeto. Sendo assim, seus artefatos podem sofrer alterações no decorrer do projeto, para se manterem íntegros no final. Os dois processos restantes são executados iterativamente, uma vez para cada funcionalidade documentada [FDD 2002].

3. Trabalhos relacionados

Os trabalhos a seguir trazem estudos sobre aplicações de diferentes técnicas de ensino e a aplicação do *Gamification* no desenvolvimento de *softwares* de aprendizado.

3.1. *GamiCAD: a gamified tutorial system for first time AutoCAD users*

Apresentado por Li, Grossman e Fitzmaurice (2012), o GamiCAD trata-se de um tutorial "gamificado" para os usuários de primeira viagem do *software* AutoCAD. Com a evolução da computação, os *softwares* e ferramentas dispõem de muitos recursos, fazendo com que o aprendizado da ferramenta torne-se difícil para os novos usuários. O excesso de tempo gasto somente para aprender sobre uma ferramenta faz com que os novos adeptos desmotivem-se. Assim, o trabalho apresenta um *software* de tutorial "gamificado" que explora o retorno audiovisual em tempo real para o usuário, o que não se viu antes em tutorial algum.

Para o desenvolvimento do tutorial, os autores criaram uma extensão para o AutoCAD utilizando o AutoCAD ObjectArx SDK, podendo-se, assim, monitorar os eventos acontecidos no próprio AutoCAD. O GamiCAD conta com dois níveis de tarefas para a aprendizagem. Para usuários principiantes, são dadas tarefas com guias de ajuda altamente estruturados. De acordo com a evolução do usuário, ele é recompensado com fases bônus do tipo "arcade".

O estudo concluiu que, utilizando o tutorial "gamificado", os usuários obtiveram resultados melhores, realizando as tarefas mais rapidamente, de maneira mais completa, e, ainda, obtiveram um nível maior de interação.

3.2. Diferentes usos do computador na educação

Apresentado por Valente (2000), o trabalho sugere uma reflexão sobre o uso do computador na educação em geral, levantando pontos sobre os diferentes usos e, mais especificamente, descrevendo os diferentes tipos de *software* educativo. No artigo, o

autor descreve que os *softwares* educativos são divididos em três categorias: os *softwares* que instruem o usuário, os *softwares* que são instruídos pelo usuário e os *softwares* com os quais os usuários manipulam a informação.

Durante o seu desenvolvimento, o trabalho apresentou uma metodologia desenvolvida pelo Instituto de Tecnologia de Massachusetts (MIT), chamada Logo. Tal metodologia trata do uso de uma linguagem de programação que permite explorar os aspectos do processo de aprendizagem, tendo tanto raízes computacionais como pedagógicas. As principais características da Logo são: exploração de atividades espaciais, fácil terminologia e capacidade de criar novos termos e procedimentos.

Apresenta-se, na conclusão do trabalho, que os mecanismos de aprendizado naturais do ser humano fazem com que as pessoas aprendam mesmo de maneira implícita, ou seja, mesmo sem que alguém diga, explicitamente, que estão aprendendo.

3.3. *DOGeometry: teaching geometry through play*

O trabalho apresentado por Wallner e Kriglstein (2012) descreve uma avaliação sobre o jogo educacional chamado DOGeometry, que tem como objetivo ensinar as noções básicas de figuras geométricas às crianças. Ao contrário de outros jogos educacionais com o mesmo objetivo, o DOGeometry aplica a geometria em objetos da natureza para chamar a atenção das crianças. O trabalho também cita que estas, geralmente, acham matemática uma matéria "chata". O estudo foi feito por meio de dados coletados durante toda a experiência dos jogadores.

Com o DOGeometry o resultado obtido não foi tão positivo, pois as crianças que participaram do teste não entenderam que os objetos do jogo eram figuras geométricas e, dessa forma, não conseguiram resolver os exercícios, com a mesma facilidade, utilizando papel e caneta.

3.4. Conclusão sobre os trabalhos relacionados

A partir dos trabalhos pesquisados, pôde-se notar que a técnica de *Gamification*, ao ser aplicada, gerou resultados positivos tanto para os desenvolvedores quanto para os usuários. No caso do trabalho DOGeometry [Wallner e Kriglstein 2012], como resultado não foi tão positivo, gerou uma preocupação com relação à explanação de que o conteúdo ali exibido faz parte de uma disciplina. Com isso, conclui-se que, ao desenvolver um jogo educativo, deve-se ter cuidado e deixar explícito de qual disciplina e matéria tratam-se os desafios. A metodologia Logo, apresentada no trabalho "Diferentes usos do computador na educação", tem em comum com este projeto o uso de uma linguagem própria para interagir com o sistema, com a diferença de que, no Logo, o aluno pode criar novos procedimentos.

4. Metodologia

Atualmente, as metodologias ágeis predominam no desenvolvimento de *software*. Para o desenvolvimento deste trabalho, foi escolhida a metodologia *Feature Driven Development* (FDD), pois esta tem características muito importantes, realização de entregas frequentes, desenvolvimento baseado na prioridade das funcionalidades e possibilidade de eventuais alterações no escopo do projeto.

Nesta seção, serão apresentadas as ferramentas previstas para o desenvolvimento deste trabalho, a definição da metodologia FDD e o detalhamento do projeto.

4.1. Ferramentas

A utilização de boas ferramentas de desenvolvimento pode aumentar drasticamente o desempenho do desenvolvedor e diminuir os tempos de entrega. O processo de seleção deve ser cauteloso, e as ferramentas disponíveis devem ser experimentadas, a fim de descobrir quais se encaixam melhor para as devidas finalidades, tornando o trabalho mais ágil e concreto.

A seguir, será apresentada uma lista contendo os nomes dos *softwares* previstos a serem utilizados durante o desenvolvimento do *software*, seguidos de uma breve explicação:

- Unity 5: plataforma principal de desenvolvimento de jogo. Será a principal ferramenta durante o desenvolvimento;
- Sony Vegas 13: plataforma que será utilizada no auxílio da criação da trilha sonora e dos efeitos sonoros;
- Microsoft Visual Studio 2015: interface de desenvolvimento utilizada para a manipulação dos códigos em C#;
- Google Chrome e Mozilla Firefox: navegadores de Internet para a execução dos testes.

4.2. Definição do modelo inicial

O processo “Desenvolver modelo inicial” é o primeiro processo do ciclo de vida de um projeto desenvolvido com FDD. Ele tem como resultado uma arquitetura inicial, chamada de *object model* (modelo de dados abrangente). Realiza-se um estudo dirigido sobre o escopo do sistema e seu contexto. Então, são realizados estudos mais detalhados sobre o domínio do negócio para cada área a ser modelada. Esse estudo pode tomar forma como um diagrama de domínio, ilustrando a estrutura física. O modelo inicial é atualizado frequentemente durante as diversas iterações do projeto, mantendo-se, assim, atualizado [FDD 2002].

4.3. Listagem de funcionalidades

É uma atividade inicial que abrange todo o projeto, a fim de identificar todas as funcionalidades que satisfaçam os requisitos do software. Esse processo tem como critérios de saída uma “Lista de funcionalidades”, podendo ser representada por um conjunto de Requisitos Funcionais (RF) e Requisitos Não Funcionais (RNF), que devem descrever as necessidades reais do negócio do ponto de vista do cliente [FDD 2002].

4.4. Planejamento por funcionalidade

É uma atividade inicial que abrange todo o projeto e tem como objetivo produzir um plano de desenvolvimento. Deve-se planejar a ordem na qual as funcionalidades serão implementadas, baseada nas dependências entre elas e também em sua complexidade. Como resultado, o plano de desenvolvimento deve ser constituído por uma tabela

descritiva das funcionalidades e seus tempos de desenvolvimento estimados [FDD 2002].

4.5. Detalhamento por funcionalidade

O processo de “Detalhamento por funcionalidade” é executado uma vez para cada funcionalidade, a fim de produzir o Pacote de Arquitetura. Nessa etapa do projeto, é criado o Diagrama de Atividade, o Diagrama de Classe e também o Diagrama Entidade Relacionamento [FDD 2002].

A Figura 5 apresenta o Diagrama de Classe previsto para o desenvolvimento do *software*.

A classe ControladorGato é responsável pela interpretação dos comandos digitados pelo usuário e pela execução das ações do terminal dentro do jogo. Essa classe é um componente do objeto de jogo “Kitten”, que corresponde ao objeto do personagem. O comando recebido é lido pela função “Interpretar”, e esta executa os comandos nas outras funções.

A classe BotaoCamera é responsável pela troca entre os dois modos de visualização do jogo, utilizando o recurso de ativar e desativar câmera.

A classe ControlaTextBox é responsável por mostrar e esconder o balão de ajuda disponível na interface. O balão mostra a lista de comandos e serve para auxiliar o jogador caso ele esqueça algum comando.

A classe CarregaFase é uma classe chamada pelo menu do jogo e tem a função de carregar o modo de jogo escolhido pelo usuário.

A classe Tutorial é responsável por toda a parte textual dos tutoriais, carregando os textos na interface do jogo. Os textos ficam armazenados em arquivos de extensão “.txt”, facilitando a alteração e possibilitando o desenvolvimento de novos tutoriais.

A classe JogoDoCaos implementa toda a mecânica do modo de jogo “Jogo do Caos”. Ela é responsável pelo sorteio dos vértices, pela geração de pontos na tela e pela implementação das funções da interface.

Todas as classes herdam as funções e os objetos do motor gráfico (*engine*).

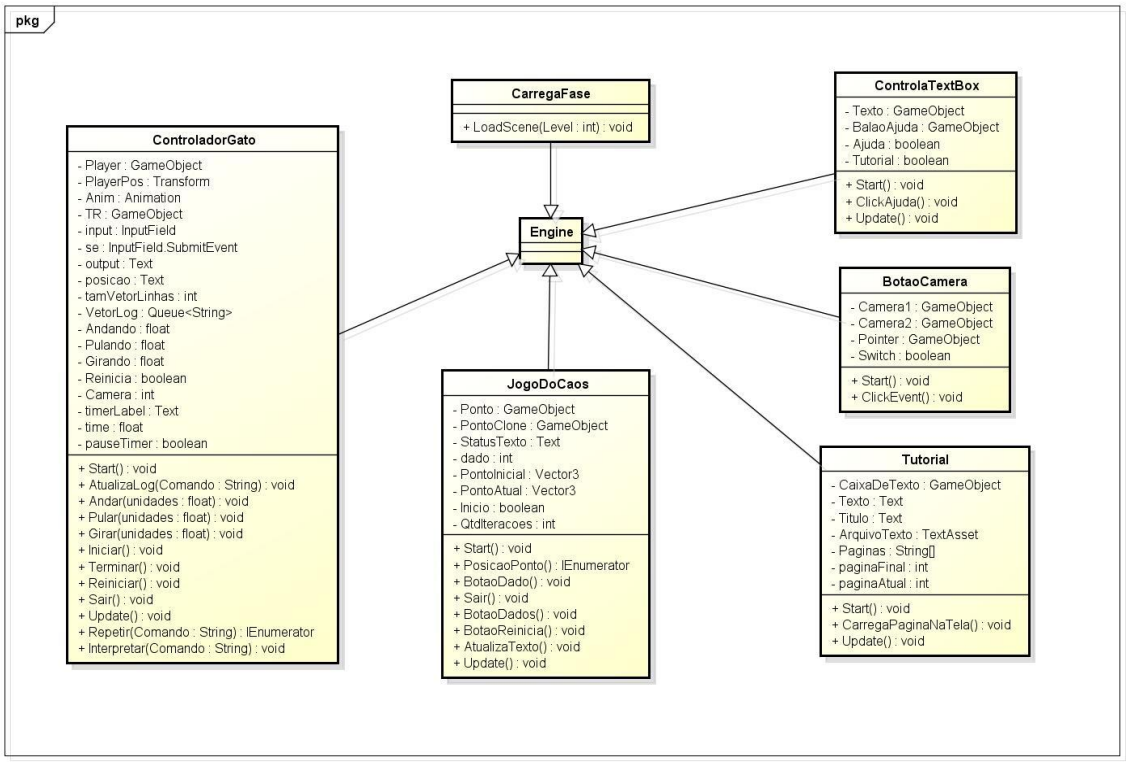


Figura 5. Diagrama de Classe

4.6. Desenvolvimento por funcionalidade

Tem como objetivo produzir (implementar um dos itens planejados e arquitetados para construir cada funcionalidade) uma função com valor para o cliente (funcionalidade), ou seja, ao final desse processo, ter-se-á uma funcionalidade implementada em código testado e inspecionado, pronto para ser entregue. O processo inicia-se com o Pacote de Arquitetura construído no processo anterior pronto [FDD 2002].

Para o início do desenvolvimento do código propriamente dito, a interface de desenvolvimento foi ajustada e organizada. Os elementos que são tratados durante o desenvolvimento de um jogo são diferentes de elementos de linguagens de baixo nível. Um projeto Unity deve ter seus elementos declarados e organizados no elemento de "hierarquia". Esse elemento da interface organiza os objetos de jogo (GameObjects), que contêm elementos chamados componentes.

A Figura 6 demonstra um fragmento de código que é implementado no método Repetir, da classe ControladorGato. Neste fragmento, o software faz um tratamento da entrada de dados do usuário e executa uma chamada à função referente ao comando "anda" do personagem. Como o processamento de um laço de execução é rápido, foi necessário o uso da função "yield return" (Figura 6, Linha 160), que faz com que a execução aguarde por um determinado período de tempo. O cálculo realizado para a espera foi baseado na multiplicação de 0.04 segundos por cada unidade que o personagem se move.

```
156                 if (cmds[j].Contains("anda"))
157                 {
158                     Interpretar(cmds[j]);
159                     string comandoAnda = cmds[j].Replace("anda", string.Empty);
160                     yield return new WaitForSeconds(0.04f * float.Parse(comandoAnda));
161                 }
```

Figura 6. Fragmento de código do método “Repetir” da classe “ControlaGato”

5. Resultados

O jogo, que leva o nome de MathCat, foi uma ideia original que surgiu tendo em mente o problema descrito na introdução deste trabalho. O jogo se ambienta em um terreno plano, que representa o plano cartesiano. Foi escolhido como personagem principal um gato que executa comandos básicos de movimentação inseridos pelo jogador para a construção de figuras geométricas. Estas figuras podem ser observadas tanto em três dimensões quanto em duas, melhorando a experiência do usuário. O jogo conta com uma fase tutorial, chamada “Tutorial Geral”, e conta com modos de jogo para o Jogo do Caos, a Curva de Koch e o Triangulo de Sierpinski. O jogo conta também com um modo livre. Este modo livre possibilita ao jogador criar qualquer desenho no plano cartesiano, instigando a criatividade e podendo também ser utilizada por um professor para ser aplicado em sala de aula, o que auxilia o dinamismo.

Toda a parte de interação com o personagem do jogo é baseada em comandos de texto. Os comandos são: “pula”, “anda”, “gira”, “repete”, “zera”, “inicia”, “termina” e “sai”. A sintaxe dos comandos no jogo é esta mesma, com exceção dos comandos “pula”, “anda”, “gira” e “repete”, os quais devem acompanhar quantas unidades serão executadas. Por exemplo, se o usuário deseja que o personagem se movimente 5 unidades para frente, ele deve digitar “anda 5”.

O comando “repete” é utilizado para repetir uma sequência de comandos. Neste caso, o usuário deve digitar o comando, seguido da quantidade de repetições desejadas e, entre parênteses, digitar os comandos que serão repetidos, separados por vírgulas. Exemplo: "repete 8 (anda 10, gira 45)". A partir do comando “repete”, o usuário evita o esforço de inserir os comandos um a um, auxiliando no processo de construção dos fractais. A Figura 7 mostra a aplicação do comando “repete” na construção de uma forma semelhante a uma estrela, utilizando o comando “repete 40 (anda 30,gira 100,anda 30,gira -50)”.

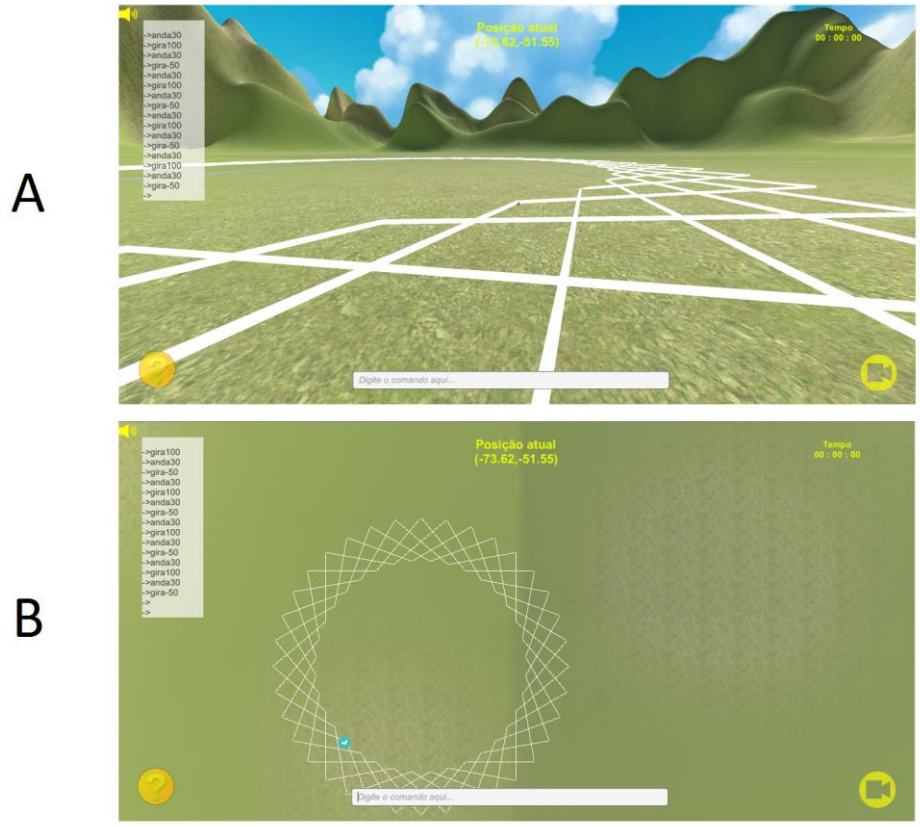


Figura 7. Interface do jogo após aplicar o comando "repete 40 (anda 30,gira 100,anda 30,gira -50)", sendo em A a visão 3D e em B a visão 2D.

Na Figura 8, é exibida a tela de interface do jogo, podendo-se notar os seguintes elementos:

- Campo de Comando (1): meio principal de interação entre o usuário e o sistema do jogo;
- Log de Comandos (2): apresenta um histórico recente dos comandos executados pelo usuário;
- Botão Troca de Câmera (3): alterna entre a câmera que mostra o personagem em uma perspectiva 3D e a câmera que mostra uma perspectiva 2D do plano cartesiano;
- Mostrador da Posição Atual (4): indica a posição atual do personagem no plano;
- Mostrador de Tempo (Timer) (5): mostra o tempo de jogo atual decorrido a partir do início do desafio;
- Botão Ajuda (6): mostra uma lista com os comandos do jogo. O usuário pode utilizar esse recurso quando esquecer de algum comando ou sua sintaxe.

Figura 8. Interface do jogo

Esta é a interface base para os modos de jogo “Modo Livre”, “Desafio Curva de Koch”, “Desafio Triângulo de Sierpinski” e “Tutorial Geral”. Para o modo de jogo “Jogo do Caos”, a interface é mais simples, pois trata-se de um jogo baseado em eventos aleatórios.

A Figura 9 mostra a interface do modo Jogo do Caos, sendo aplicado de acordo com as regras descritas na Seção 2.4.3.

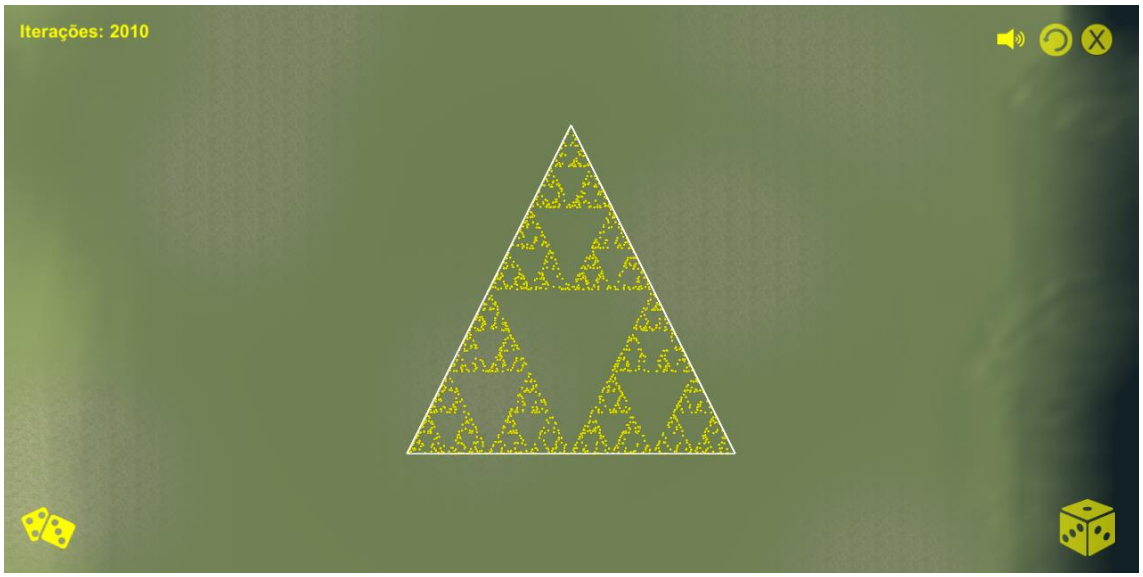


Figura 9. Interface do modo de jogo “Jogo do Caos”, após a aplicação de 2010 iterações

6. Conclusão

Conforme as pesquisas realizadas para o desenvolvimento deste trabalho, pode-se concluir que, para fazer um jogo, não basta somente saber programar, mas também ter criatividade para explorar a interação e a imersão do jogador por meio de elementos audiovisuais e responsividade. Ao final do desenvolvimento do presente trabalho, os resultados superaram as expectativas, pois a ferramenta desenvolvida se mostrou versátil, abrangendo não somente a geometria fractal, mas também parte da geometria plana.

Sobre o aprendizado da ferramenta Unity, é possível encontrar grande parte dos recursos e tutoriais na Internet. Existe uma grande comunidade de desenvolvedores e entusiastas que trocam informações por meio dos fóruns. A Unity também conta com uma documentação muito bem-escrita. O desenvolvimento tornou-se muito intuitivo, ajudando o desenvolvedor a alcançar os objetivos desejados.

Segundo o professor de matemática Rodrigo Fioravanti Pereira, docente do Centro Universitário Franciscano, o jogo alcança e amplia seu objetivo, pois proporciona o desenvolvimento de tópicos mais gerais da geometria, como áreas, volumes, ângulos, posições relativas entre retas, entre outros. A proposta inicial privilegiava a abordagem plana dos fractais, mas, pelo que foi visto até aqui, o projeto tem potencial para desenvolver o conteúdo também em seu enfoque espacial. Assim, para o futuro, espera-se que um eixo norteador, baseado na teoria dos fractais, ramifique-se em tópicos que envolvam outros conteúdos da geometria plana e espacial, de tal maneira que a jogabilidade mescle-se, cada vez mais, com o conteúdo trabalhado, fazendo com que o usuário identifique, cada vez menos, se está jogando ou aprendendo geometria.

Entende-se também que o jogo desenvolvido pode trazer momentos de diversão e ensino. Futuramente, poderão ser acrescentadas funcionalidades, como contagem de pontos, e novos desafios poderão ser propostos, além de nova jogabilidade, novos comandos etc. Poderá ser implementado, ainda, um servidor fixo e ativo, viabilizando a implementação de sistema de cadastro do usuário, o que também tornaria possíveis o arquivamento e a análise dos dados evolutivos dos jogadores.

7. Referências

Alves, F. (2015) “Gamification: como criar experiências de aprendizagem engajadoras”, 2ª edição. DVS Editora.

Barbosa, R. M. (2002) “Descobrindo a Geometria Fractal para a sala de aula”, Coleção Tendências em Educação Matemática, Editora Autêntica.

FDD. (2002) “Feature Driven Development Processes”, http://www.featuredrivendevelopment.com/files/fddprocessesA4.pdf, May.

Davidson & Associates (1993) "Math Blaster - Episode One". Fresno, CA.

Janos, M. (2008) “Geometria Fractal”, Editora Ciência Moderna.

Kawashima, K. (1998) “Análise da Aplicação de Fractais à Obtenção de Texturas Paramétricas Contínuas em Computação Gráfica”, Dissertação submetida como

requisito parcial para a obtenção do grau de Mestre em Ciência da Computação, UFRGS, Porto Alegre, Janeiro.

Khronos Group (2016) “WebGL - OpenGL ES 2.0 for the Web”, https://www.khronos.org/webgl/, April.

Lemes, D. O. (2009) “Games independentes: fundamentos metodológicos para criação, planejamento e desenvolvimento de jogos digitais”, PUC-SP.

Li, W., Grossman, T. and Fitzmaurice, G. (2012) “GamiCAD: A Gamified Tutorial System For First Time AutoCAD Users”, Autodesk Research.

Lopes, C. e Pantaleão, C. H. Z. (2016) “Algumas Técnicas de Análise de Imagens utilizando Fractais”, Figura de Exemplo do processo de construção da Curva de Koch, http://www.inf.ufsc.br/~visao/2000/fractais/, abril.

Pacievitch, T. “História da Matemática”. http://www.infoescola.com/matematica/historia-da-matematica/, abril.

Perucia, A., Berthem, A., Bertschinger, G. e Castro, R. R. (2007) “Desenvolvimento de Jogos Eletrônicos – Teoria e Pratica”, 2ª edição. Editora Novatec.

Pressman, R. S. (2010) “Software Engineering a Practitioner’s Approach”, Sixth Edition, McGraw-Hill Education.

Spinadel, V. W., Perera, J. G. e Perera, J. H. (1993) “Geometria Fractal”, Editora Nueva Librería.

Tarouco, L. M. R., Roland, L. C., Fabre, M. J. M. e Konrath, M. L. P. (2004) “Jogos Educacionais”, CINTED-UFRGS.

The Learning Company (1999) "Coelho sabido - 2º Ano".

Unity (2016) “Unity - Game Engine”, https://unity3d.com/pt, May.

Valente, J. A. (2000) “Diferentes usos do Computador na Educação”, Núcleo de Informática Aplicada a Educação - NIED/UNICAMP.

Wallner, G. and Kriglstein, S. (2012) “DOGeometry: Teaching Geometry Through Play”, Fun and Games, Tolouse, França.

Wikipédia (2016) “Triângulo de Sierpinski”, Figura de Exemplo das iterações do Triangulo de Sierpinski, https://pt.wikipedia.org/wiki/Tri%C3%A2ngulo_de_Sierpinski, maio.

APÊNDICE A – Interfaces de jogo

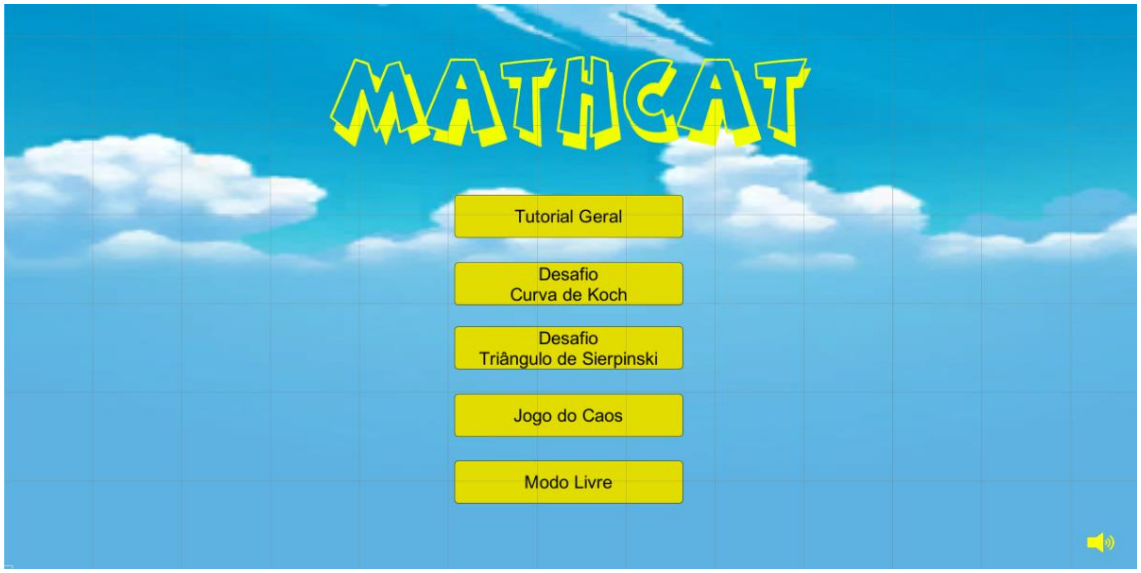


Figura 1. Menu inicial de jogo

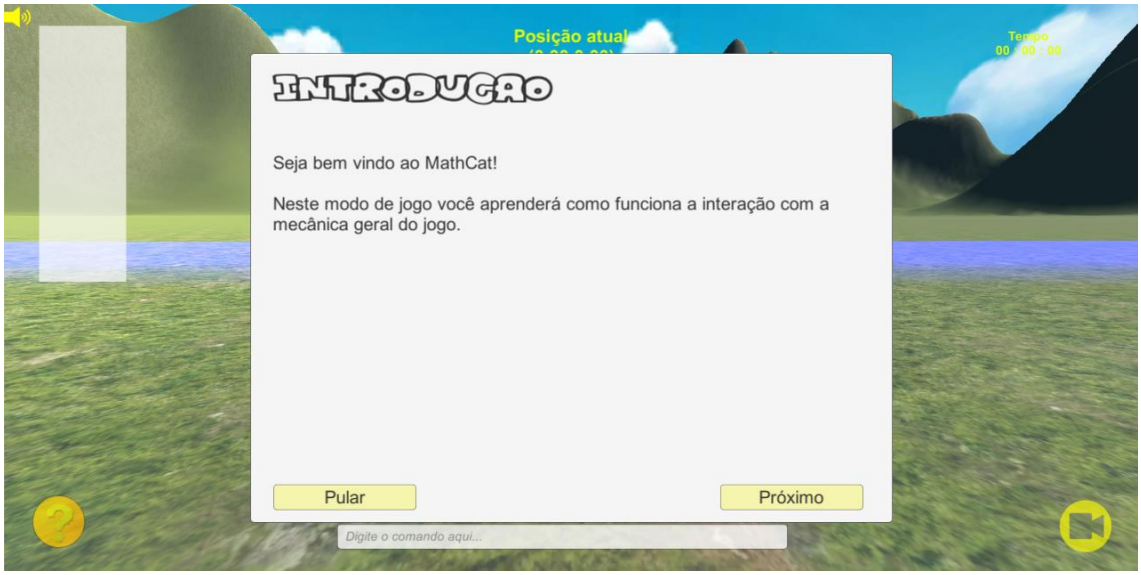


Figura 2. Interface do modo de jogo "Tutorial Geral", mostrando as dicas para o usuário.

APÊNDICE B – Implementação do comando “repete”

```
1 reference
IEnumerator Repetir(string Comando)
{

    // Divide o comando em numero de repetições e sequência de comandos
    string[] str = Comando.Split('(', ')');
    Debug.Log("" + str);

    // Interpreta quantas vezes repete
    string comandoRepete = str[0];
    comandoRepete = comandoRepete.Replace("repete", string.Empty);
    Debug.Log("" + comandoRepete);

    // Divide os comandos que serão executados
    string[] cmds = str[1].Split(',');


    // Aqui executa
    float vezes = float.Parse(comandoRepete);
    for (float i = 0; i < vezes; i++)  // Repete X vezes os comandos
        for (int j = 0; j < cmds.Length; j++)
        { // Executa cada comando por vez

            if (cmds[j].Contains("anda"))
            {
                Interpretar(cmds[j]);
                string comandoAnda = cmds[j].Replace("anda", string.Empty);
                yield return new WaitForSeconds(0.04f * float.Parse(comandoAnda));
            }

            // Personagem pula
            else if (cmds[j].Contains("pula"))
            {
                Interpretar(cmds[j]);
                string comandoPula = cmds[j].Replace("pula", string.Empty);
                yield return new WaitForSeconds(0.04f * float.Parse(comandoPula));
            }

            // Personagem pula
            else if (cmds[j].Contains("gira"))
            {
                Interpretar(cmds[j]);
                yield return new WaitForSeconds(0.02f);
            }

            else if (cmds[j].Contains("repete"))
            {
                Interpretar(cmds[j]);  // Envia o comando para a função que interpreta
                string comandoRpt = cmds[j].Replace("repete", string.Empty);
                yield return new WaitForSeconds(0.02f * float.Parse(comandoRpt));
            }

            else Interpretar(cmds[j]);  // Envia o comando para a função que interpreta
        }
}
```

Figura 1. Implementação da função do comando “repete”.

APÊNDICE C – Documentos do software (FDD)

Na tabela 1, é apresentado o planejamento, demonstrando a ordem na qual as funcionalidades foram desenvolvidas.

Tabela 1. Planejamento por funcionalidade

Ordem	Funcionalidade	Tempo para desenvolvimento
1	RNF04 - Identificar e interpretar os comandos	3 dias
2	RF06 - Modo livre	15 dias
3	RF02 - Tutorial geral de funcionamento	10 dias
4	RF03 - Desafio Curva de Koch	10 dias
5	RF04 - Desafio Triângulo de Sierpinski	10 dias
6	RF05 - Jogo do Caos	10 dias
7	RNF06 - Efeitos sonoros	5 dias
8	RNF07 - Trilha sonora	15 dias
9	RF01 - Menu principal de jogo	5 dias

A Figura 1 representa o modelo abrangente, apresentando-o por meio de um diagrama de domínio.

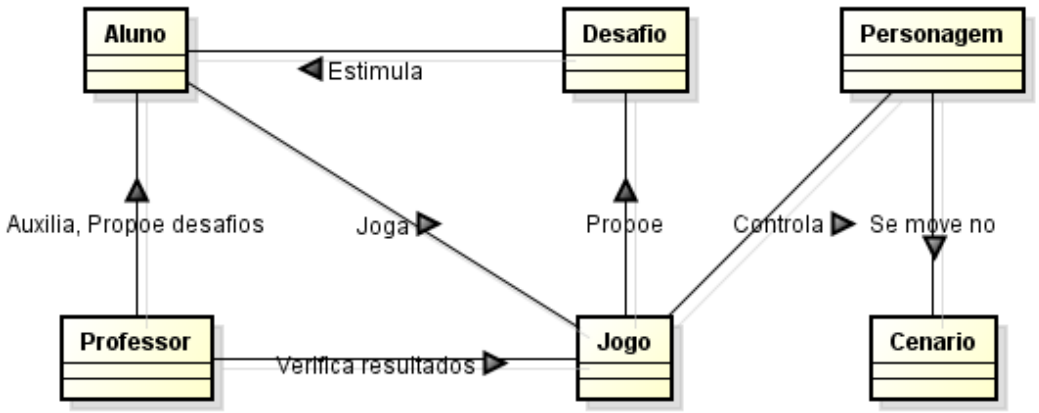


Figura 1. Diagrama de domínio

Os requisitos funcionais previstos para este trabalho são:

RF01 – Iniciar Menu principal de jogo

- RF01.1 – Fornecer opção para Tutorial geral de funcionamento
- RF01.2 – Fornecer opção para Desafio Curva de Koch
- RF01.3 – Fornecer opção para Desafio Triangulo de Sierpinski
- RF01.4 – Fornecer opção para Jogo do caos
- RF01.5 – Fornecer opção de silenciar jogo

RF02 – Jogar Tutorial geral de funcionamento

- RF02.1 – Fornecer dicas de uso geral

RF03 – Jogar Desafio Curva de Koch

- RF03.1 – Fornecer dicas de construção para a Curva de Koch
- RF03.2 – Exibir pontuação

RF04 – Jogar Desafio Triangulo de Sierpinski

- RF04.1 – Fornecer dicas de construção para o Triangulo de Sierpinski
- RF04.2 – Exibir pontuação

RF05 – Jogar Jogo do caos

RF06 – Jogar Modo livre

Os requisitos não funcionais são:

RNF01 – Utilizar a ferramenta Unity 5

RNF02 – Utilizar a linguagem de programação JavaScript

RNF03 – Utilizar a API WebGL

RNF04 – Identificar e interpretar os comandos

RNF05 – Executar o software através do navegador

RNF06 – Efeitos sonoros

RNF07 – Trilha sonora