Two Kingdoms: Relato de Desenvolvimento de um Jogo Utilizando Godot Engine

Gustavo Guerreiro
Instituto Federal Catarinense – Campus Blumenau
gustavo.guerreiro.ifc@gmail.com

Lucas Vargas
Instituto Federal Catarinense – Campus Blumenau
lucasvargas27@hotmail.com

Igor Zafriel Schmidt
Instituto Federal Catarinense – Campus Blumenau
igorschmidt52@gmail.com

Ricardo de la Rocha Ladeira
Instituto Federal Catarinense – Campus Blumenau
ricardo.ladeira@ifc.edu.br

ABSTRACT

This paper aims to explain the game development experience creating 'Two Kingdoms' an Action RPG with an 8-way movement, top-down camera, focused on exploration, and a multi-stage adventure while explaining the functionalities of the used engine. The different abilities and knowledge necessary to develop a game and how they were learned will also be included, like coding, math, physics, music, art, and storyline creation. The article addresses game creation experience, such as the choosing process of the engine, the tools used for art and music creation, and the explanation of some functions provided by Godot Engine for the development of the game presented priorly.

KEYWORDS

Godot; Programming; Game Development;

1 INTRODUÇÃO

Um dos meios mais comuns de entretenimento na sociedade contemporânea são jogos eletrônicos, na verdade, essa indústria é a que mais fatura dentro da área de entretenimento, ultrapassando a indústria do cinema e da música [1]. O Brasil é o décimo terceiro maior mercado de *videogames* do mundo, e o maior da América Latina, movimentando aproximadamente 5,6 bilhões de reais ao ano [2]. O público alcançado pelos jogos eletrônicos é muito abrangente, incluindo diferentes gêneros, etnias e idades [3].

Com isso em mente criou-se Two Kingdoms, um jogo com câmera no estilo *top-down* para um jogador dividido em fases com enredo simples e linear, onde o objetivo é concluir os diferentes níveis para alcançar o fim do jogo e a conclusão da história.

A partir disso, este artigo pretende detalhar a experiência do desenvolvimento do jogo, o motivo pelo qual a engine Godot foi escolhida, como se deu a aquisição de informações para fazê-lo, a programação do jogo, os processos para criação de músicas e artes e a explicação das áreas do conhecimento envolvidas na produção, como matemática e física.

Outro trabalho que aborda a experiência e o processo de criação de um jogo é o artigo feito por Otto et al [4], que se aprofunda nas mecânicas e características de um jogo criado na biblioteca Arcade do Python chamado Bombanimal, servindo como inspiração no processo de desenvolvimento deste trabalho, que além dos temas semelhantes também traz uma comparação de engines e a inserção de música.

2 MATERIAIS E MÉTODOS

Para a produção do jogo utilizou-se a Engine de jogos Godot por ser gratuita e dispor de uma linguagem semelhante a Python [5] já conhecida pelos integrantes do grupo. A criação dos *sprites* foram feitas utilizando o aplicativo pago Aseprite[1] pela sua interface amigável e funções que facilitam seu uso, no quesito das músicas, foram desenvolvidas em *websites* de criação musical chamados BeepBox[2] e JummBox [3] e por fim, no quesito de ferramentas, utilizamos o GitHub[4] como repositório e forma de compartilhamento para os arquivos e mídia do jogo.

Quanto ao aprendizado, usou-se como fonte de informação além da própria documentação do GDScript[6] (Linguagem de programação do Godot) guias e vídeos disponíveis na *internet*, dos quais inicialmente originaram diversos protótipos de teste com objetivo de ganhar experiência com o Godot e ter expectativas mais realistas quanto ao produto final. Por fim foi criado um protótipo evolucionário com os conhecimentos adquiridos, com pequenas funções sendo incrementadas gradativamente e o uso de um GDD (disponível no repositório do GitHub) para nortear o desenvolvimento do projeto.

3 DESENVOLVIMENTO

3.1 Escolha da Engine

Como conhecimento prévio, havia o básico de programação em Python, entendimento de elementos como variáveis, operações, condicionais, funções, etc. Por isso, preferiu-se optar por alguma *engine* que usasse essa linguagem ou outra semelhante. Por isso inicialmente foram cogitadas as bibliotecas Pygame e Arcade para Python, contudo, estas apresentavam poucas funcionalidades para se alcançar o que era planejado, já que se baseava somente em linhas de código e não contava com uma interface e funções mais avançadas como outras ferramentas de criação de jogos como Unity ou GameMaker Studio 2, então decidiu-se pesquisar por outras ferramentas.

[1] https://www.aseprite.org/
[2] https://www.beepbox.com
[3] https://jummbus.bitbucket.io
[4] https://github.com/NikoronBR/Two-Kingdowns

A GameMaker Studio 2 possui uma interface amigável e funções como criação de *sprites* interna, contudo, o seu custo era de 169,99 reais, um preço maior do que o grupo estava disposto a pagar. Outra engine avaliada foi a Unreal Engine 4, mas esta demanda mais poder de processamento que outras como Unity e Godot e os efeitos gráficos e funções presentes nela tem foco em jogos tridimensionais, além disso, apresenta uma interface mais complexa de se dominar do que outras como a Unity, possuindo uma curva de aprendizado maior, e como o grupo possuía 10 meses para finalizar o projeto, uma engine mais fácil de se aprender seria o desejável. Chegou-se a duas outras opções para desenvolver o jogo: Unity e Godot. Ambas são gratuitas, têm alta capacidade de fazer jogos 2D, possuem interfaces simples e fáceis de se aprender e possibilitam o desenvolvimento de jogos completos com qualidade.

O ponto decisivo para a escolha foi a linguagem de programação utilizada, Unity utiliza-se da linguagem C# que os integrantes do grupo não conhecem, já Godot Engine usa GDScript, uma linguagem própria fortemente baseada em Python, a qual os integrantes já tinham certa familiaridade. Além disso, Godot possui código aberto, qualquer pessoa tem acesso ao código da ferramenta e pode criar versões modificadas com funcionalidades extras que podem ser adicionadas ao original. Godot continua sendo constantemente atualizado com cada vez mais elementos, muitos deles sugeridos pela comunidade que é bem ativa como pode ser visto no *subreddit* da *engine* [7], onde tanto a comunidade dá sugestões como os desenvolvedores respondem dúvidas e postam sobre atualizações no programa. Por fim, Godot também pode ser usado com diferentes linguagens: GDScrpit, C# e C++. A programação também pode ser feita usando VisualScript um modo de adicionar *scripts* visuais para aqueles que não tem familiaridade com programação.

Atributos	Unity	Unreal	Pygame/ Arcade	Godot	Game Maker
Gratuito	Sim	Sim	Sim	Sim	Não
Interface Fácil	Sim	Sim	Não	Sim	Sim
Linguagem Principal	C#	C++	Python	GD Script	GML

Tabela 1: Tabela comparativa entre *engines*

A Tabela 1 faz uma comparação entre as *engines* e explicita o fato de que o Godot é gratuito, apresenta uma interface fácil e usa GDScript, que é semelhante a Python.

Existem outros trabalhos em que engines são comparadas, como o de Mishra e Shrawankar [8], em que é feita uma comparação completa entre os motores, contudo, sem a citação de Godot ou alguma ferramente baseada em Python, e o trabalho de Cavalcante e Pereira [9], em que são comparadas Godot, Unity e Phaser, neste trabalho também chegou-se à conclusão de que o Godot era a mais adequada naquele contexto por ser simples e gratuita, ter código aberto e diversas ferramentas que possibilitam resultados satisfatórios.

3.2 Funcionamento da Godot Engine

3.2.1 *Nodes e Scenes*

Na engine Godot os principais elementos são *scenes* e *nodes*, *nodes* são objetos que carregam funções e desempenham diversos papéis dentro do jogo, alguns mostram imagens enquanto outros lidam com animações e física. A *scene* é formada por um grupo de *nodes* organizados de forma hierárquica onde *nodes* podem herdar a outros, uma *scene* pode representar um ''nível'' e quando rodamos o jogo é uma *scene* e seus vários *nodes* que são carregados, podem também representar um objeto específico como um jogador com *nodes* que compõem suas funções, assim, uma *scene* ''nível'' poderia ter uma instância da *scene* ''jogador'' dentro dela [10].

Para exemplificar pode-se imaginar que se crie uma *scene* contendo um gramado de fundo, o gramado é desenhado por dois *nodes*, o *sprite* da grama e o *TileMap*, este é um *node* pai do *sprite* que permite que o desenvolvedor "desenhe" o plano de fundo segurando o botão do *mouse* e o arrastando, assim inserindo vários *sprites* um ao lado do outro no fundo. Em cima do gramado pode haver uma casa com colisão, ou seja, um corpo não pode atravessá-la, tanto a imagem da casa como a colisão são *nodes* (*Sprite* e *CollisionShape2D* respectivamente), esses elementos estão dentro de outro que representa a junção da imagem com a colisão para formar a casa (*StaticBody2D*), tanto o *TileMap* do gramado e o *StaticBody2D* da casa estão dentro de um *node* principal chamado de *Node2D*, que engloba todos os elementos de jogos 2D em uma *scene*, para testar os objetos o desenvolvedor pode iniciá-la, se precisar testar algum elemento separado, terá que criar outro nível, ou *scene*, para testá-lo, e é nessa estrutura que a linguagem se organiza, uma árvore com *nodes* interagindo entre si dentro de um nível, como pode-se observar na Figura 1.

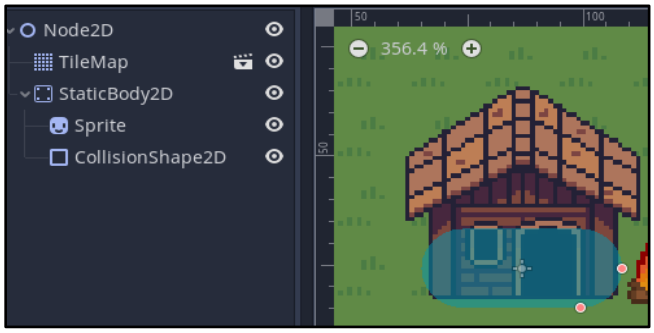

Figura 1: Representação de uma casa no Godot

3.2.2 *Programação*

A programação no Godot é feita na forma de *scripts* que são adicionados e atribuem características aos *nodes*, como a vida, velocidade e aceleração do jogador, a área que a câmera irá abranger e como irá se comportar (ficar fixa, seguir o jogador, se movimentar quando o jogador passar da borda, etc.) e como alguns elementos irão interagir com outros (jogador perder vida ao entrar em contato com inimigos ou inimigos sofrerem dano ao interagirem com a espada do jogador, por exemplo).

É possível criar funções que são ativadas quando algum evento específico ocorre no jogo por meio de sinais, que são enviados ao código e podem modificar coisas no jogo assim que o evento acontece, por exemplo, existe um item de chave no chão, assim que o jogador passa por cima é enviado um sinal para o código que faz com que a chave desapareça do chão e o contador de chaves no canto da tela aumente. O Godot também possibilita herança de objetos, ou seja, um *script* pode herdar os dados de outro, como variáveis e funções, permitindo uma programação orientada a objeto.

A linguagem de programação usada no Godot é semelhante a Python, na sintaxe, onde a endentação correta é necessária para o código funcionar, nos laços de repetição que funcionam da mesma forma e nas variáveis e funções que são definidas de formas parecidas, dentre diversas outras semelhanças que fazem com que aqueles que já têm certa familiaridade com Python tenham facilidade em fazer os *scripts* para Godot, como pode-se perceber na seção de perguntas frequentes na documentação da engine.

3.3 Música

Músicas desempenham um papel importante em jogos eletrônicos, representando funções metafóricas como sensação de espaço, caracterização da atmosfera e ambiente em que os eventos se passam (campos verdejantes e alegres ou castelos perigosos e assustadores, por exemplo), também tem função metonímica, contribuindo para a narração do jogo, indicando quando um inimigo se aproxima, quando ocorre uma batalha contra um chefe etc [11].

Para o processo de criação musical, utilizou-se de *websites* gratuitos com foco em criação de músicas do tipo *8-bits* (tipo de música que imita as limitações sonoras de *videogames* da década de 1980 como Nintendo Entertainment System e Sega Master System), os *sites* utilizados foram BeepBox e JummBox, o segundo trata-se de uma versão modificada do primeiro, ambos permitem a criação de diferentes camadas da música, como melodia, harmonia e ritmo, possibilitam que o criador escolha o tipo de instrumento utilizado e coloque as notas em uma grade, a posição, afinamento e comprimento das notas pode ser alterado por meio de uma interface gráfica.

A adição de músicas no Godot é assim como para arte, por meio de *nodes*, utilizando o *node AudioStreamPlayer* podemos inserir um arquivo de áudio ogg ou wav e configurar como se deseja que se comporte no jogo, se a música irá tocar em loop, se irá tocar só uma vez, ou se irá parar de tocar quando um inimigo específico morrer, para este último, por exemplo, poderíamos fazer com que o *node AudioStreamPlayer* com o áudio desejado fosse filho de tal inimigo, assim, pelo fator hierárquico, quando o inimigo morresse o *node* filho contendo o áudio iria ter suas funções suspendidas.

3.4 Arte

A arte dentro do contexto de um jogo engloba tudo aquilo que é visível ao jogador, é por ela que se identifica o mundo, os personagens, os inimigos e todos os outros elementos presentes, a arte é de certo modo uma representação visual das várias linhas de código que compõem o que chamamos de "jogador" ou "inimigo", deste modo podemos por meio dela construir uma narrativa e situar o jogador no universo que ele se encontra.

Para criar a parte gráfica do jogo optou-se pela *Pixel Art,* uma forma de arte digital onde os *pixels* (menor ponto de uma imagem digital) são individualmente colocados sobre uma tela a fim de montar uma figura completa. Foi escolhido a *Pixel Art* levando em conta a inexperiência do grupo e a sua simplicidade quando comparada com os modos tradicionais de arte, sendo possível criar desenhos decentes mesmo com pouca experiência na área. A principal ferramenta utilizada para criação dos desenhos ou *sprites* foi o software Aseprite, e nele se originaram o design do jogador, inimigos, objetos, hud e *tileset* que podem ser vistos nas Figuras 2 e 3.

Figura 2: Elementos do Jogo

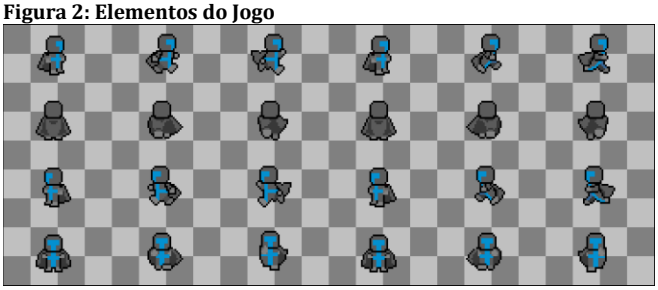

Figura 3: *Sprites* do jogador

Para a inserção de arte gráfica o Godot conta com diversos *nodes* com funções específicas para manipulação de *spritesheets* e realização de animações, o *node TileMap* por exemplo permite a composição do cenário do jogo por meio de *tilesets*, o *node sprite* permite a inserção de uma imagem para usos gerais, o *node AnimationPlayer* trabalha com *spritesheets* que demonstram uma animação, permitindo definir o tempo e categorizar cada parte dessa animação para ser posteriormente referenciada no *script,* assim é possível definir quando e onde a animação será ativada e quais efeitos ela terá no jogo.

3.5 O JOGO

3.5.1 *Enredo*

O jogo se passa em um mundo chamado Soulwin, onde existem dois grandes reinos, o reino da tecnologia e o reino da magia. Por muitos anos os reinos conviveram em paz, porém, motivados pela aquisição de mais territórios, riquezas e poder, os reinos entraram em uma duradoura guerra que transformou grande parte de Soulwin em ruínas e fez de seus habitantes criaturas hostis. Em meio a devastação surge uma jovem de nome Agnes que tem como missão restabelecer a ordem e a paz de seu mundo, derrotando os inimigos e superando os desafios que aparecem em seu caminho.

3.5.2 *Jogabilidade*

A jogabilidade foi feita inspirada em The Legend of Zelda no estilo *action RPG* em 2D com uma câmera do tipo *top-down*. O jogador pode se mover em oito direções usando o teclado do computador e controle de Xbox One ou Playstation 4. Conforme o passar do jogo a personagem adquire mais opções, inicialmente é mais lenta e não pode atacar, com o passar do jogo, consegue armadura e espada que aumentam a sua velocidade e resistência além de permitir que ataque os inimigos.

3.5.3 *Matemática e Física Envolvida*

Alguns elementos do jogo contém conceitos matemáticos e físicos na sua realização, a movimentação do jogador por exemplo é realizada por meio de um plano cartesiano e vetores, suponha-se que o jogador tenha pressionado o botão para se movimentar a direita, o valor 1 seria alocado na variável vetorial x, esse valor seria multiplicado com a velocidade máxima do jogador e uma aceleração iria determinar o tempo que levaria para que o jogador chegasse nessa velocidade máxima, se o jogador soltasse o botão, a variável vetorial x assumiria o valor 0 e um valor de frenagem iria determinar o tempo que o jogador levaria para parar.

Na Figura 4 abaixo vemos um exemplo da aplicação de conceitos físicos e matemáticos como vetores, velocidade, aceleração e fricção para fazer a movimentação do personagem.

```
var input_vector = Vector2.ZERO
input_vector.x = Input.get_action_strength("ui_right")
- Input.get_action_strength("ui_left")
input_vector.y = Input.get_action_strength("ui_down")
- Input.get_action_strength("ui_up")
input_vector = input_vector.normalized()

if input_vector != Vector2.ZERO:
    velocity = velocity.move_toward(input_vector * MAX_SPEED,
    ACCELERATION * delta)

else:
    animationState.travel("Idle")
    velocity = velocity.move_toward(Vector2.ZERO,
    FRICTION * delta)
```

Figura 4: Elementos do Jogo

3.5.4 *Níveis*

O jogo se passa em um mundo com elementos mágicos e tecnológicos, o primeiro nível é introdutório, onde o jogador inicia sem armadura e pode explorar uma área pacífica com grama e árvores, conforme segue o caminho se depara com um portal que o leva para o segundo nível, um labirinto tecnológico com obstáculos e desafios com serras pelos quais o jogador precisa passar, além disso inimigos mágicos também aparecem no caminho, existem dois deles, um persegue o jogador e o outro atira projéteis mágicos. Por fim, chega ao final onde encontra uma espada e um portal que leva até um chefe que o persegue até ficar cansado e entrar em um estado vulnerável, nesse momento abre margem para ser atacado e então voltar a perseguir Agnes, após receber 3 golpes o chefe é derrotado.

4 CONSIDERAÇÕES FINAIS

O uso da *engine* Godot se mostrou uma experiência positiva para o grupo devido ao conhecimento prévio em Python, a gratuidade do programa e as funcionalidades simples de se aprender e que ao mesmo tempo geram resultados satisfatórios, se mostrando viável para iniciantes.

A produção do jogo contribuiu para diversas áreas do conhecimento como arte, música, criação de enredo, matemática, física, programação e engenharia de *software*, além do planejamento. Enriquecendo as habilidades dos integrantes do grupo e se mostrando um bom meio de aprendizado.

O desenvolvimento do jogo não demandou muitos recursos financeiros, contudo, necessitou de tempo e planejamento para aprender o que e como precisa ser feito, desde a leitura da documentação e guias até a criação de protótipos para teste e desenvolvimento da versão final.

5 REFERÊNCIAS

[1] J. Harada, "Que indústria fatura mais: do cinema, das músicas ou dos games?" https://super.abril.com.br/mundo-estranho/que-industria-fatura-mais-do-cinema-da-musica-ou-dos-games(accessed Dez. 18, 2020)

[2] N. Larghi, "Brasil é o 13º maior mercado de games do mundo e o maior da América Latina" https://valorinveste.globo.com/objetivo/empreenda-se/noticia/2019/07/30/brasil-e-o-13o-maior-mercado-de-games-do-mundo-e-o-maior-da-america-latina.ghtml(acessed Dez. 18, 2020)

[3] U. Ritterfeld and R. Weber, "Video Games for Entertainment and Education", Set. 2005.

[4] V. U. Otto et al, "Apresentação das Mecânicas de um jogo desenvolvido com Arcade" Set, 2020.

[5] https://www.python.org/

[6] https://docs.godotengine.org/en/stable/index.html

[7] https://www.reddit.com/r/godot/

[8] P. Mishra and U. Shrawankar, "Comparison between Famous Game Engines and Eminent Games" Jan, 2016.

[9] C. H. L. Cavalcante and M. L. A. Pereira "Comparativo entre Game Engines como Etapa Inicial para o Desenvolvimento de um Jogo de Educação Financeira" Jun, 2018.

[10] M. Nations, "Godot's Node System, Part 3: Engine Comparisons." https://willnationsdev.wordpress.com/2018/04/07/godots-node-system-part-3-engine-comparisons/(acessed Dez. 18,2020)

[11] Z. N. Whalen, "Play Along: Video Game Music as Metaphor and Metonymy', 2004.3