Desenvolvimento de jogos na plataforma Godot

Vitor Poltronieri da Silva[1], Igor Yepes[2]

[1] Curso Técnico Integrado em Informática do Instituto Federal de Educação, Ciência e Tecnologia Farroupilha (IFFar) – Frederico Westphalen – RS – Brasil

[2] Instituto Federal de Educação, Ciência e Tecnologia Farroupilha (IFFar) – Frederico Westphalen – RS – Brasil

vitorpoltronieri.irai@gmail.com, igor.yepes@iffarroupilha.edu.br

Abstract. *People with little experience in game development are usually apprehensive of the common sense that "games need complex graphics, they need to be done by big companies to be well done, they need great multidisciplinary teams and programmers with advanced knowledge in development." Given this, the objective of this project is to influence people who wish to produce games, to release their creativity and to express it without fear, using a tool for developing computer games (Godot game engine) with great usability and productivity, which uses the programming language for games GDScript, based on Python. As a result of the work, it was possible to develop a 2D game with retro look, simple but nice graphics, and good gameplay.*

Keywords: *Godot, Game development, Retro game, 2D game.*

Resumo. *Normalmente, pessoas com pouca experiência em desenvolvimento de games têm receio do senso comum de que "jogos precisam de gráficos complexos, necessitam ser feitos por empresas grandes para serem bem feitos, precisam de grandes equipes multidisciplinares e de programadores com conhecimento avançado em desenvolvimento". Visto isso, o objetivo deste projeto é influenciar pessoas que desejem produzir games, a usufruir da sua criatividade e expressá-la sem medo, utilizando uma ferramenta para desenvolvimento de jogos computacionais (Godot) com uma ótima usabilidade e produtividade, que usa a linguagem de programação para jogos GDScript, baseada em Python. Como resultado do trabalho, foi possível desenvolver um jogo 2D com visual retrô, de gráficos simples, mas agradáveis, e boa jogabilidade.*

Palavras-chave*: Godot, Desenvolvimento de jogos, Jogo retrô, Jogo 2D.*

1. Introdução

Os jogos computacionais estão inseridos no cotidiano das pessoas há bastante tempo. Com seu surgimento na década de 50, inicialmente, tiveram o propósito exclusivo de entretenimento, e, por estarem em constante evolução, graças ao advento de equipamentos com cada vez melhor capacidade de processamento, estão sendo explorados para os mais diversos fins, abrangendo desde a área educacional, para ensino de tópicos específicos em diversas disciplinas, até treinamentos, mediante uso de simuladores na área médica, industrial, comercial, militar entre outras. Entretanto, seu

maior impacto comercial continua na área de entretenimento, com uma perspectiva financeira de U$ 1,4 bilhão até o ano 2022 (Diniz, 2017).

Atualmente a criação de jogos está se difundindo entre pequenos desenvolvedores, deixando de ser exclusividade de grandes empresas. Hoje em dia, é possível desenvolver um jogo individualmente ou em pequenas equipes e comercializá-lo utilizando recursos da Internet.

2. Game engine Godot

Godot é um motor de desenvolvimento de jogos (*Game Engine*) compatível com várias plataformas 2D e 3D, lançado como software de código aberto sob a licença MIT. Foi inicialmente desenvolvido para várias empresas na América Latina antes de seu lançamento público. O ambiente de desenvolvimento poder ser executado no Windows, macOS, Linux, BSD e Haiku (ambos de 32 e 64 bits) e pode criar jogos destinados a plataformas PC, console, dispositivos móveis e web (Linietsky, 2016).

A linguagem básica de programação usada na *engine* Godot é o GDScript. É através dessa linguagem que são chamadas as funções, programada a velocidade e a gravidade, entre tantos outros pontos essenciais para o desenvolvimento prático do jogo.

Os desenvolvedores de Godot afirmaram que muitas linguagens alternativas de script de terceiros (a saber, Lua, Python e Squirrel) foram testadas antes de decidir que o uso de uma linguagem customizada permitia uma otimização e integração de editor superiores (Linietsky & Manzur, 2016). Assim, o Godot incorpora o GDScript, linguagem própria, inspirada em Python, porém otimizada para trabalhar com velocidade na *engine*.

O Python é uma linguagem de programação de alto nível, interpretada, de script, imperativa, orientada a objetos, funcional, de tipagem dinâmica e forte. Foi lançada por Guido van Rossum em 1991, e, atualmente, possui um modelo de desenvolvimento comunitário, aberto e gerenciado pela organização sem fins lucrativos *Python Software Foundation.* Apesar de várias partes da linguagem possuírem padrões e especificações formais, a linguagem como um todo não é formalmente especificada. O padrão de fato é a implementação CPython (Python, 2008). Uma de suas principais características é permitir a fácil leitura do código e exigir poucas linhas de codificação se comparado ao mesmo programa em outras linguagens. Devido às suas características, ela é principalmente utilizada para processamento de textos, dados científicos e criação de CGIs (*Common Gateway Interface*) para páginas dinâmicas para a web (Python, 2008).

O GDScript foi desenvolvido levando em conta diversas características do Python, principalmente no que se refere à clareza do código, sua facilidade de manutenção e sua curva de aprendizado, bastante interessante, principalmente, para quem está iniciando na área de programação.

3. O jogo: R.O.B.Y (Relayed Omnifunctional Battle Yovern) - Adventure of an Misplaced Robot

O jogo é do tipo aventura, onde o personagem atravessa um mapa para descobrir segredos, derrotar chefes e superar diversos desafios, exigindo atenção e destreza do usuário. Seu visual foi inspirado no estilo dos jogos dos anos 90, podendo ser

categorizado dentro do estilo retrô, tendo como base jogos como Super Mario Bros®, MegaMan®, Castlevania®, Yoshi's Island® e Final Fantasy®.

O jogo é desenvolvido em 2D, pela simplicidade e por ser um estilo com uma fácil curva de aprendizado no que se refere à parte física e mecânica, exigindo, neste caso, o envolvimento de menos profissionais de diversas áreas para sua construção. Jogos mais elaborados, necessitam atuação de profissionais de diversas áreas, além da área de programação. Jogos 2D, são mais viáveis para programadores individuais, podendo utilizar repositórios de imagens livres para uso com os personagens e cenários.

A história do jogo se passa no ano 3482, na Terra. Neste ano, um cientista, chamado de Magus, cria uma máquina que pode enviar objetos pelo espaço-tempo, para outras dimensões. Para testar a máquina, ele envia o robô de batalha R.O.B.Y.. Porém, devido a um erro de cálculo, Roby acaba indo para uma dimensão alternativa, fora de rota. Agora, ele tem que fazer de tudo para retornar à sua dimensão original.

O mesmo se passa em um plano Bidimensional (2D), sendo que o personagem precisa andar em uma direção padrão (para a direita) na busca de atingir o final de cada fase.

4. Materiais e métodos

Na *engine* Godot, todas as entidades são criadas em um sistema parental, onde um construtor é filho de outro. Dessa maneira, vem a facilidade do desenvolvimento, pois é possível controlar vários construtores de maneiras diferentes.

Todos os blocos trabalham em conjunto, aplicando gravidade, elasticidade, entre outros. Assim, um construtor pode enviar um sinal para outro, e, dessa maneira, produzir as entidades.

A *engine* permite o desenvolvimento multiplataforma. Isto é, o jogo pode ser exportado para diferentes sistemas operacionais, *desktop* e *mobile*. O presente projeto foi parcialmente produzido em um iMac, e a parte final foi produzida em um *desktop* PC, utilizando o sistema operacional Windows. Isso ressalta as características de portabilidade da *engine*.

Dentro do Godot, existem diversas ferramentas para o desenvolvimento de *games*. Este trabalho utilizou os métodos de desenvolvimento básicos da *engine*, os quais são explicados a seguir.

4.1. Nodes

Nodes são blocos de construção fundamentais para criar um jogo. Um *node* pode executar uma variedade de funções especializadas. No entanto, qualquer *node* fornecido sempre possui os seguintes atributos (Linietsky & Manzur, 2014):

- Tem um nome;
- Tem propriedades editáveis;
- Pode receber um retorno de chamada para processar cada quadro;
- Pode ser estendido (para ter mais funções).

E, principalmente:

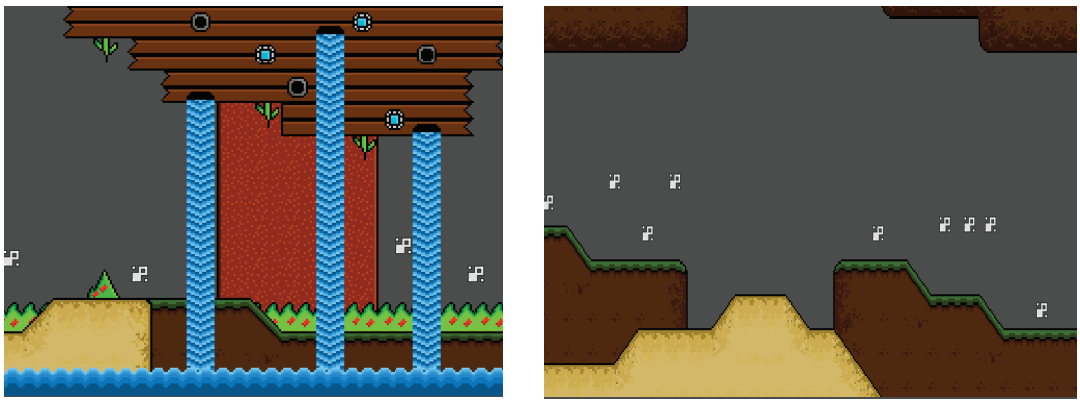
- Pode ser adicionado a outros *nodes* como filho. Quando organizados dessa maneira, os *nodes* se tornam uma árvore.

Em Godot, a capacidade de organizar os *nodes* dessa maneira cria uma ferramenta poderosa para organizar projetos. Como *nodes* diferentes têm funções diferentes, combiná-los permite a criação de funções mais complexas.

Cada *node* possui um tipo e uma função. No projeto, foram usados o *Node2D, Tilemaps, AudioStreamPlayer2D, CollisionObject2D, Area2D, KinematicBody2D, Camera2D, Sprite & AnimatedSprite* e *Control*, os quais serão brevemente explicados nos tópicos seguintes.

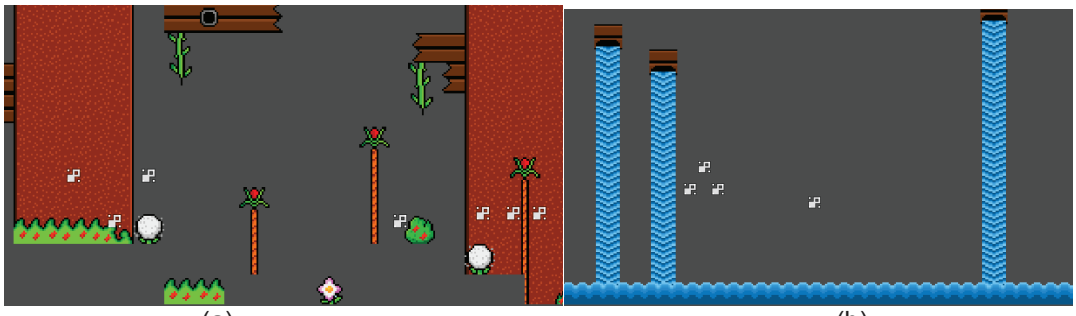
***Node2D*:** é um objeto de jogo 2D, com posição, rotação e escala. Todos os *nodes* 2D e *sprites* de física 2D são filhos do *Node2D*. É usado como um nó pai para mover e dimensionar em um projeto 2D (Linietsky & Manzur, 2014). Neste trabalho, foi utilizado como controlador principal e organizador parental dos *nodes*.

***TileMaps*:** *node* para mapas baseados em blocos 2D. *Tilemaps* usam um *TileSet* (pacote de blocos) que contém uma lista de blocos (texturas com colisão opcional, navegação e/ou formas de oclusão) que são usados para criar mapas baseados em grade (Linietsky & Manzur, 2014). Foram usados um total de 7 *Tilemaps*, na seguinte ordem: *Background1* (blocos de fundo, madeiras, etc), *Background2* (árvores, arbustos, etc), *Solids* (plataformas sólidas), *Ground* (blocos de terra que colidem com o jogador), *Sand* (blocos de areia que colidem com o jogador) *Waterfall* (cachoeiras que ficam a frente da tela) e *Water* (água na parte de baixo da tela), como pode ser observado nas figuras a seguir (Figs. 1 e 2).



(a) (b)

Fig. 1 - (a) Exemplo de uso de todos os *Tilemaps* juntos. (b) Exemplo de *tilemaps* de chão e teto (*Ground* e *Sand*), separados.



(a) (b)

Fig. 2 - (a) Exemplo de *Tilemaps* de Fundo (*Background1* e 2), separados. (b) Exemplo de uso de *Tilemaps* de frente (*Water* e *Waterfall*).

***AudioStreamPlayer2D*:** reproduz áudio, possuindo tipos suportados, sendo eles o .ogg e .wav. O som amortece (vai reduzindo) com a distância do centro da tela.
Usado em efeitos sonoros, como pulo e ao coletar uma moeda, e músicas, como a música tema executada durante o jogo.

***CollisionObject2D*:** é a classe base para objetos físicos 2D. Pode conter qualquer número de *Shape* 2D (formas) de colisão 2D. Cada forma deve ser atribuída a um proprietário de forma (*CollisionShape2D* ou *CollisionPolygon2D*) (Linietsky & Manzur, 2014). O *CollisionObject2D* pode ter qualquer número de proprietários de formas. Os proprietários de formas não são nodes e não aparecem no editor, mas podem ser acessados por meio do código. Foi usado em todas as entidades trabalhadas, para definir formas e colisões, como mostrado na figura 3.

Fig. 3 - Exemplo de uso do *ColisionObject2D* (quadrado azul acima da entidade), como colisão da entidade e colisão da área, para enviar sinal de contato.

***Area2D*:** área que detecta colisões, entidades que entram, saem e que estão dentro da colisão da mesma. Está no personagem para detectar a área de dano, nas moedas, em transições de tela, e em inimigos. Na figura 4, há um exemplo de código de uma Area2D, para remover uma moeda que foi coletada.

```
1  extends Area2D
2
3  func _on_Moeda_body_entered(body):
4      get_node("CoinSprite").play("coin_smoke")
5      get_node("collis").queue_free()
6      get_node("som").play()
7      yield(get_node("CoinSprite"), "animation_finished")
8      queue_free()
9      pass
10
```

Fig. 4 - Exemplo de código de Area2D, usado em uma moeda.

***KinematicBody2D* (Corpos Cinemáticos):** são tipos especiais de corpos que devem ser controlados pelo usuário. Eles não são afetados pela física - para outros tipos de corpos, como um personagem ou um corpo rígido, eles são o mesmo que um corpo estático (Linietsky & Manzur, 2014). Eles possuem, no entanto, dois usos principais: Movimento Simulado e *Kinematic Characters*. - o Movimento Simulado ocorre quando esses corpos são movidos manualmente, seja por meio de código ou de um *AnimationPlayer*, a física irá calcular automaticamente uma estimativa de sua velocidade linear e angular. Isso os torna muito úteis para movimentar plataformas ou outros objetos (como uma porta, uma ponte que se abre, etc). Um exemplo de movimento simulado são os inimigos, que não utilizam nenhum comando, como é mostrado na Figura 5.

Fig. 5 - Inimigo usado no jogo. Possui física implantada no seu pulo, quando identifica uma entidade acima dele.

Já o *Kinematic Characters* (*KinematicBody2D*) também possui uma API para mover objetos (os métodos *move_and_collide* e *move_and_slide*) enquanto realiza testes de colisão. Isso os torna realmente úteis para implementar personagens que colidam contra um mundo, mas isso não exige física avançada. O *Kinematic Character*, foi utilizado para o personagem principal, como é visto nas figuras 6 (a e b).

(a)

```
if Input.is_action_pressed("ui_right"):
    motion.x = SPEED
    $PSprite.flip_h = false
    $PSprite.play("run")
elif Input.is_action_pressed("ui_left"):
    motion.x = -SPEED
    $PSprite.flip_h = true
    $PSprite.play("run")
else:
    motion.x = 0
    $PSprite.play("stopped")
```

(b)

Fig. 6 - (a) Personagem principal do jogo, o que o usuário controla. (b) Exemplo de código usado para movimentar o personagem no mapa.

***Camera2D*:** *node* da câmera para cenas 2D. Ele força a tela (camada atual) a rolar seguindo este *node*. Isso torna mais fácil (e mais rápido) programar cenas roláveis do que alterar manualmente a posição dos *nodes* baseados em *CanvasItem* (tudo o que é visível no jogo) (Linietsky & Manzur, 2014). Está como base para a movimentação do personagem pelo cenário.

***Sprite & AnimatedSprite*:** o *sprite* é o *node* que exibe uma textura 2D. A textura exibida pode ser uma região de uma textura de atlas maior ou um quadro de uma animação de folha de *sprite* (Linietsky & Manzur, 2014). Usado como gráfico base de *tilesets*, entre outros. O *Animated Sprite* é o *node* que pode usar várias texturas para animação. Foi usado nas entidades que se movimentam, como moedas, personagens, inimigos, e outros, como é mostrado na figura 7.

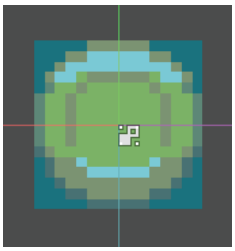


Fig. 7 - Gráfico da moeda, com *Animated Sprite*.

***Control* (Controle):** todos os nós da interface do usuário são herdados do controle. Apresenta âncoras e margens para adaptar sua posição e tamanho ao pai (Linietsky & Manzur, 2014). Foi usado na interface para calcular a quantidade de moedas, o status da arma, e as interfaces para entrar no jogo, como é apresentado na figura 8.

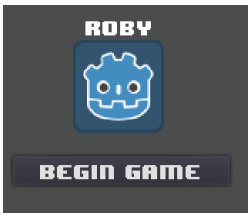


Fig. 8 - Exemplo de UI dentro do jogo.

4.2. Cenas

Uma cena é composta de um grupo de *nodes* organizados hierarquicamente (em forma de árvore). Além disso, segundo Linietsky & Manzur (2014), uma cena:

- sempre tem apenas um node raiz;

- pode ser salva em disco e carregado de volta;

- pode ser instanciada.

Executar um jogo significa executar uma cena. Um projeto pode conter várias cenas, mas para o jogo começar, uma delas deve ser selecionada como a cena principal.

Basicamente, o editor Godot é um editor de cena. Tem muitas ferramentas para editar cenas 2D e 3D, bem como interfaces de usuário, mas o editor é baseado no conceito de editar uma cena e os nodes que a compõem (Linietsky & Manzur, 2014).

5. Resultados e considerações finais

O jogo desenvolvido atingiu as expectativas esperadas, necessitando pouco processamento para ser executado e rodando em capacidade máxima em todas as plataformas testadas (Windows e Mac).

Contudo, foram encontrados *bugs* nas entidades inimigas. Sobre isso, aconteciam erros no código que matava os inimigos, o que foi detectado como um problema da *engine*, não do código escrito, após consultas aos fóruns especializados.

Apesar da *engine* ter uma interface de fácil compreensão, integrar os *nodes* não é tão simples, pois precisa de um conhecimento prévio de como os mesmos funcionam.

O jogo desenvolvido (Fig. 9) foi exposto na VI Mostra Regional de Ciências IFFar Campus FW, tendo sido avaliado por professores e visitantes. Recebeu um

feedback positivo, somente com algumas críticas pela sua dificuldade, pois poucas pessoas passavam da metade da fase construída. Essa dificuldade se deu em decorrência das posições dos inimigos e pela difícil mecânica de movimento desenvolvida. Esse problema foi resolvido após os testes, para que o entretenimento seja mais leve e para que os usuários possam usufruir mais da jogabilidade e da história do jogo.

Fig. 9 - Exemplo de tela do jogo finalizado.

Como trabalho futuro, planeja-se realizar o desenvolvimento de novas fases entre novas dimensões, integrar mapas e chefes no final de cada uma, desenvolver novos personagens e oponentes, adicionar caixas de diálogo e animações para explicar a história, entre outros. Após isso, se planeja publicar o jogo para venda nas plataformas *Steam, Google Play* e *Apple Store*, nos sistemas operacionais Windows, Android e MacOS a um preço acessível, sendo que, na sua versão final, ele possuirá diálogos em inglês, para facilitar a capilarização e acessibilidade dos Usuários.

Referências

DINIZ, Ana Carolina. Games: número de desenvolvedores de jogos cresce de 43 para 300. O Globo, 2017, Acessado em: 12/09/2018, Disponível em: https://oglobo.globo.com/economia/emprego/games-numero-de-desenvolvedores-de-jogos-cresce-de-43-para-300-21919137

LINIETSKY, Juan; MANZUR, Ariel. Godot Engine Last Documentation, 2014, Acessado em: 02/09/2018, Disponível em: http://docs.godotengine.org/en/3.0.

LINIETSKY, Juan; MANZUR, Ariel. Godot game engine. Godot, 2016. Disponível em: <https://web.archive.org/web/20160206055951/http://godotengine.org/projects/godot-engine/wiki/GDScript>. Acessado em: 20/09/2018.

LINIETSKY, Juan. "Godot 2.0: Talking with the Creator". 2016. Disponível em: <http://80.lv/articles/godot2-interview/>. Acessado em: 20/09/2018,

PYTHON Software Foundation. Python 3.0 Release. Python.org, 2008, Acessado em: 02/09/2018, Disponível em: https://www.python.org/download/releases/3.0/.