UNIVERSIDADE FEDERAL DO CEARÁ

CAMPUS QUIXADÁ

BACHARELADO EM ENGENHARIA DE SOFTWARE

GELDERSON BEZERRA ALVES

ESTUDO COMPARATIVO ENTRE ENGINES DE DESENVOLVIMENTO DE JOGOS 2D

QUIXADÁ

2021

GELDERSON BEZERRA ALVES

ESTUDO COMPARATIVO ENTRE ENGINES DE DESENVOLVIMENTO DE JOGOS 2D

Monografia apresentada no curso de Engenharia de Software da Universidade Federal do Ceará, como requisito parcial à obtenção do título de bacharel em Engenharia de Software. Área de concentração: Computação.

Orientadora: Prof. Dra. Paulyne Matthews Jucá

QUIXADÁ

2021

Dados Internacionais de Catalogação na Publicação
Universidade Federal do Ceará
Biblioteca Universitária
Gerada automaticamente pelo módulo Catalog, mediante os dados fornecidos pelo(a) autor(a)

A479e Alves, Gelderson Bezerra.
Estudo comparativo entre engines de desenvolvimento de jogos 2D / Gelderson Bezerra Alves. – 2021.
43 f. : il. color.

Trabalho de Conclusão de Curso (graduação) – Universidade Federal do Ceará, Campus de Quixadá, Curso de Engenharia de Software, Quixadá, 2021.
Orientação: Profa. Dra. Paulyne Matthews Jucá.

1. Jogos-Desenvolvimento. 2. Unity. 3. Design de jogos. I. Título.

CDD 005.1

GELDERSON BEZERRA ALVES

ESTUDO COMPARATIVO ENTRE ENGINES DE DESENVOLVIMENTO DE JOGOS 2D

Monografia apresentada no curso de Engenharia de Software da Universidade Federal do Ceará, como requisito parcial à obtenção do título de bacharel em Engenharia de Software. Área de concentração: Computação.

Aprovada em: ___/___/___

BANCA EXAMINADORA

Prof. Dra. Paulyne Matthews Jucá (Orientadora)
Universidade Federal do Ceará – UFC

Prof. Dr. Jefferson de Carvalho Silva
Universidade Federal do Ceará - UFC

Prof. Ma. Diana Braga Nogueira
Universidade Federal do Ceará - UFC

À minha Mãe.

AGRADECIMENTOS

A minha mãe, pela educação que me proporcionou desde o início da minha vida, pelo amor, incentivo e apoio incondicional.

Aos meu amigos e namorada, especificamente Arthur Antunes, Matheus Rios e Antonio Reis por terem me ajudado e motivado a seguir em frente.

A todos que direta ou indiretamente fizeram parte da minha formação, muito obrigado a todos.

"Quem olha para fora sonha, quem olha para dentro desperta."

(Carl Jung)

RESUMO

O ambiente atual da internet proporciona que a criação de jogos independentese torne uma realidade no mercado de desenvolvimento de *games*. Com isso, diversas ferramentas para desenvolver jogos de forma ágil surgiram na comunidade. Escolher a melhor *engine* para desenvolver um jogo dessa natureza nem sempre é fácil. Este trabalho visa a comparação entre duas das *engines* mais populares atualmente para o desenvolvimento de jogos independentes 2D, *Unity* e *Construct* 3. O objetivo principal é comparar diversos aspectos em relação a fase de desenvolvimento do jogo escolhido em cada *engine*. Os parâmetros de comparação são de natureza unicamente técnicas, pra que o desenvolvedor consiga entender exatamente as características que *engine* apresenta. A criação do mesmo jogo em ambas ferramentas, é o passo crucial para de fato realizar essa comparação e conseguir obter resultados sólidos. Os resultados da comparação evidenciam que, enquanto a *Unity* dá mais poder de desenvolvimento para o desenvolvedor ao custo da complexidade da ferramenta, a *Construct* 3 é mais simples de desenvolver um jogo 2D.

Palavras-chave: Jogos-Desenvolvimento. Unity. Design de jogos.

ABSTRACT

The current internet environment allows the creation of independent games to be a reality in the *games* development market. With that, several tools to develop games in an agile way emerged in the community. Choosing the best *engine* to develop a game of this nature is not always easy. This work aims to compare two of the most popular *engines* today for the development of independent 2D games *Unity* and *Construct* 3. The main objective is to compare different aspects in relation to the development phase of the game chosen in each *engine*. The comparison parameters are of a technical nature only, so that the developer can understand exactly the characteristics that *engine* presents. The creation of the same game in both tools, is the crucial step to actually make this comparison and obtain solid results. The results of the comparison show that, while *Unity* gives more developer power to the developer at the cost of the complexity of the tool, *Construct* 3 is simpler to develop a 2D game.

Keywords: Games-Development. Unity. Game design.

LISTA DE FIGURAS

LISTA DE QUADROS

SUMÁRIO

1 INTRODUÇÃO

Engines de jogos são softwares complexos, estes produtos tem o objetivo bem claro de facilitar o desenvolvimento de games. Em geral, o desenvolvimento de jogos envolve um ambiente multidisciplinar que abrange áreas distintas de conhecimento. As *engines* tem a função de dar suporte a estas diferentes partes, transformando-as em ferramentas composta por vários módulos com objetivos distintos que culminam em um produto final com qualidade e abrangência (SILVA; MARTINS, 2015). Estas áreas podem variar entre: inteligência artificial, suporte a multiplataforma, renderização de imagens, gráfico 3d, composições sonoras, animação, físicas realistas, entre outras (FERNANDES, 2018). Algumas *engines* mais complexas dão até mesmo suporte para desenvolvimento de ambientes de realidade virtual (FERNANDES et al., 2015). O mercado apresenta várias *engines* e cada uma com suas particularidades, vantagens e desvantagens.

Diante da complexidade das *engines*, realizar a escolha de uma *engine* específica ocorre de maneira cuidadosa e com isso é feita uma análise nos pontos a serem considerados do projeto. Conceitos como satisfação de uso, facilidade de aprendizado, suporte da comunidade também devem ser analisados. Estes aspectos são extremamente importantes, pois é comum que projetos de desenvolvimento de games demorem bastante, às vezes até anos (O'HAGAN; COLEMAN; O'CONNOR, 2014), e a qualidade nesses aspectos impacta o tempo de desenvolvimento e o sucesso do projeto como um todo. Com isso, este trabalho realiza uma análise e comparação dos diversos aspectos essenciais na escolha de uma *engine* para desenvolvimento de um jogo 2D.

Conforme a ideia de interdisciplinaridade e complexidade do desenvolvimento de um jogo, o time de desenvolvimento também necessita de aptidões em diferentes áreas. Alguns times são menores e podem até ser compostos por apenas um membro, mas a ideia ainda se mantém, pois, em geral nesses casos, um mesmo membro trabalhará em mais de uma vertente diferente. A equipe de desenvolvimento também tem a opção de desenvolver a própria *engine* ou pode usar um software já existente, que pode ser gratuito, necessitar de licença e termos para fins financeiros (O'HAGAN; COLEMAN; O'CONNOR, 2014). Esta decisão é de extrema importância e impacta fortemente o desenvolvimento em geral do jogo e decorrente disso, o produto final.

Assim, este trabalho se propõe a comparar as duas *engines*, Unity e Construct 3 para o desenvolvimento de jogos 2D em dispositivos moveis. Essas *engines* foram escolhidas pois são muito populares dentro da comunidade de criação de novos jogos independentes, como está exposto no site *itch.io*.

Visando analisar e comparar as duas ferramentas, neste trabalho será desenvolvido um jogo em ambas as plataformas. O jogo focará em ser abrangente em aspectos comuns a jogos 2D, pois características essenciais poderão ser analisados e comparados. O trabalho terá uma coleta de dados de desenvolvimento para uma observação das *engines* e será realizada a comparação dos aspectos gerais do processo de desenvolvimento em cada plataforma. Além disso, métricas objetivas em relação à aspectos técnicos de cada *engine* também são avaliados e verificados.

Na conclusão, serão apresentados e discutidos os resultados obtidos sobre qual ferramenta mais se sobressaiu baseado nas métricas de avaliação e nas observações notadas durante todo o processo em cada uma das ferramentas. Tendo em vista que o desenvolvimento de jogos e os mesmos, possuem uma grande variação, serão expostos cenários específicos do meio de desenvolvimento de jogos 2D e o resultado de qual *engine* mais se adéqua para este caso.

Os demais capítulos estão ordenados do seguinte modo: no Capítulo 2 encontra-se as análises e comparações deste com trabalhos relacionados; no Capítulo 3 está descrita toda a fundamentação teórica essencial no entendimento deste trabalho; no Capítulo 4 é exposto o método deste trabalho e seus devidos arranjos; no Capítulo 5 está presente o *Game Design Document* do jogo proposto e desenvolvido; o Capítulo 6 descreve os resultados alcançados e a comparação de cada métrica levantada; por fim, o capítulo 7 faz considerações sobre um panorama geral do trabalho e também indagações sobre trabalhos futuros partindo deste.

1.1 Objetivo Geral

O objetivo geral desse trabalho é realizar a comparação das game *engines Unity* e *Construct 3* através do desenvolvimento de um jogo 2D .

1.1.1 Objetivos específicos

- Definir as características técnicas das *engines* propostas, envolvendo desde aspectos técnicos à características subjetivas de cada uma que serão utilizadas para realizar a

comparação;

- Desenvolver um jogo 2D em cada uma das *engines* propostas;
- Avaliar as *engines* propostas com base nas métricas de desenvolvimento do jogo.
- Definir os cenários onde cada *engine* se destaca com base na experiencia adquirida durante a execução do trabalho.

2 TRABALHOS RELACIONADOS

Neste capítulo, serão listados e discutidos os trabalhos relacionados. As semelhanças e diferenças serão apontadas, assim como as contribuições que os trabalhos apresentados fazem a este. Os trabalho relacionados são (ŠMÍD,), (SILVA, 2018) e (CAVALCANTE; PEREIRA, 2018).

2.1 Comparison of unity and unreal engine (ŠMÍD,)

O trabalho de (ŠMÍD,) apresenta uma comparação entre as *game engines Unity e Unreal*. É proposta uma comparação de maneira genérica em relação ao desenvolvimento de jogos digitais e a escolha dessas duas plataformas de desenvolvimento foram baseadas nas suas grandes popularidades. A comparação é feita com base nos diversos aspectos específicos de cada ferramenta e em características de desenvolvimento de jogos como um todo. Neste trabalho, a *Unity* também é analisada. Ele apresenta métricas consistentes para uma análise comparativa focada no desempenho do jogo após o desenvolvimento. De maneira objetiva, sua análise faz a comparação das ferramentas com foco maior nos seguintes aspectos:

- Velocidade dos *frames* com aumento da complexidade do labirinto para PC, notebook, Android e GearVR
- Velocidade dos *frames* com Ajuste de resolução
- Velocidade de desempenho geral com baixa configuração de renderização
- Tamanho final dos projetos em *Megabytes*
- Qualidade Visual
- Opiniões subjetivas do desenvolvedor

Além disso, o autor mostra os pontos positivos e negativos na maioria dos pontos analisados e separa as considerações gerais para cada *engine*, com grande destaque sempre para as categorias de performance do jogo. Ambos os trabalhos comparam a *engine* Unity com outra ferramenta, mas as analises possuíram algumas diferenças. O trabalho de (ŠMÍD,) faz uma análise genérica e com foco em performance, enquanto que neste trabalho características como tempo de desenvolvimento, suporte da comunidade, curva de aprendizado e satisfação de uso serão analisadas.

2.2 Análise de ferramentas e desenvolvimento de jogo para treinamento de paratletas (SILVA, 2018)

Em (SILVA, 2018) é realizado um estudo comparativo entre *engines* de jogos com foco no levantamento de requisitos necessários para o desenvolvimento de um jogo a fim de treinar paratletas com auxilio de um Kinect, um sensor de captura de movimentos corpóreos. Inicialmente é realizado um estudo exploratório de diversas *engines* de desenvolvimento de jogos como a Unity, Unreal, Cryengine e Lumberyard. Ele extrai as características que irão facilitar o desenvolvimento do jogo proposto de cada uma das *engine* estudadas. Ao fim desse estudo, ele escolhe somente a Unity para desenvolver o jogo em questão. Além disso, ele seleciona características comuns a todas as *engines* estudadas que são:

- Sistema de script;
- Curva de aprendizado;
- Licenciamento e custo;
- Documentação e suporte;
- Heterogeneidade;
- Compatibilidade;
- Qualidade visual.

A realização da comparação das *engines* é feita baseado em estudos teóricos com uma profundidade no que há na literatura de cada uma das *engines* de desenvolvimento. A principal diferença entre o trabalho de (SILVA, 2018) e o presente trabalho é que a comparação das *engines* de desenvolvimento será realizada com mais propriedades já que o jogo proposto neste trabalho será desenvolvido em ambas *engines*, *Unity e Construct 3*.

2.3 Comparativo entre game *engines* como etapa inicial para o desenvolvimento de um jogo de educação financeira (CAVALCANTE; PEREIRA, 2018)

O trabalho apresentado em (CAVALCANTE; PEREIRA, 2018) consiste em comparar *engines* atuais de desenvolvimento de jogos a fim de escolher uma que mais se adéque para a realização do desenvolvimento de um jogo infantil de educação financeira.

Inicialmente ela destaca a importância da educação financeira para a população e em seguida apresenta um dado que apenas 35% da população brasileira pode ser considerada alfabetizada financeiramente. Para chegar resultado, foi feita uma pesquisa com um grupo

amostral de 150 mil pessoas aproximadamente. É também dado ênfase em como a gameficação de processos, através de jogos, facilita a compreensão de um assunto especifico. Com disso, ela justifica que com um jogo de educação financeira é possível divulgar conteúdo que ensine de forma indireta as sobre questões financeiras de forma a auxiliar o público infantil. Com essa justificativa, o principal objetivo do trabalho é comparar as *engines* de desenvolvimento de jogos Unity, Godot e Phaser para escolher a que mais se adéqua no desenvolvimento de um jogo de educação financeira para o publico infantil.

As *engines* foram escolhidas baseadas na qualidade da documentação e de exemplos práticao, além de serem focados em desenvolvimento de jogos 2D. O trabalho apresenta um método direto e objetivo, submetendo cada *engine* a três etapas de avaliação, que são:

- Análise da Interface e funcionalidades;
- Desenvolvimento de um jogo;
- Vantagens e Desvantagens.

Após a avaliação das etapas com cada *engine* utilizada, ele destaca que a *engine* de desenvolvimento Godot foi a que teve mais pontos positivos por mais leve e ter a possibilidade de exportar jogos para várias plataformas, além de sua interface e linguagem de programação bem amigáveis. A principal diferença do atual trabalho em relação ao trabalho de (CAVALCANTE; PEREIRA, 2018) é a finalidade da comparação, já que o intuito do trabalho atual é fazer um estudo comparativo entre às *engines* Unity e Construct 3 pois apresentam mais facilidades em desenvolvimento de jogos de plataforma 2D (GAMEFROMSCRATCH.COM,) (UNITY,). Fatores mais específicos em fase de desenvolvimento serão comparados para justificar ações que melhor são realizadas em cada uma.

2.4 Comparação deste trabalho com trabalhos relacionados

O trabalho presente tem o objetivo de comparar duas *engines* de desenvolvimento de jogos, Unity e Construct 3, com o objetivo de destacar as semelhanças e diferenças entre elas e também como cada uma se encaixa partindo da perspectiva do desenvolvedor. A principal diferença entre o trabalho proposto e os demais são as métricas de comparação utilizadas. As métricas de avaliação escolhidas neste trabalho são objetivas em relação ao desenvolvimento de jogos, então, o conhecimento necessário de manuseio de ambas *engines* se torna primordial. A escolha de métricas mais objetivas que pudessem ser apuradas na fase de desenvolvimento, foi motivada por sugestão da banca e do orientador do trabalho.

No Quadro 1, são destacadas as principais diferenças entre o presente trabalho com os trabalhos relacionados. São levados em consideração a finalidade principal de cada trabalho e as *engines* de desenvolvimento de jogos utilizados por cada um.

Quadro 1 – Comparações deste trabalho com os trabalhos relacionados.

Métricas de Comparação	**Trabalho Proposto**	**(ŠMÍD, 2017)**	**(SILVA, 2018)**	**(CAVALCANTE; PEREIRA, 2018)**
Principal finalidade	Comparar engines de desenvolvimento de jogos 2D.	Avaliação da performance de um jogo desenvolvido em duas engines distintas.	Avaliar engines e selecionar a que melhor atende os requisitos de desenvolvimento para um jogo de treinamento de paratletas.	Avaliar engines e selecionar a que melhor atende os requisitos de desenvolvimento para um jogo de educação financeira.
Engines Utilizadas	Unity e Construct 3	Unity e Unreal	Unity, Unreal, Cryengine e Lumberyard.	Unity, Godot e Phaser
Métricas de Comparação	Sistema de movimento 2d Top Down, Sitema de luz liga/desliga, Sistema de IA simples de perseguição, Colisão, Animação, Sons, Importação de Assets, Criação de leveis e Transição de leveis.	Plataformas, editor, formatos importados, renderização, animação, física, ferramentas cinematográficas, terreno e folhagem, interface de usuário, inteligência artificial e otimização.	Sistema de script, curva de aprendizado, licenciamento e custos, documentação e suporte, heterogeneidade, compatibilidade, qualidade visual, loja de assets e ecossistema.	Interface, funcionalidade das engines, plataformas, licenciamento e custos, documentação e suporte.

Fonte: Elaborado pelo autor.

3 FUNDAMENTAÇÃO TEÓRICA

Neste capítulo, é abordada toda a teoria necessária pra compreensão e desenvolvimento do trabalho. Nas Seções 3.1 e 3.2 , estão expostos os conceitos gerais de jogos, destacando jogos em segunda dimensão e jogos mobile. Na Seção 3.3, apresenta-se toda a parte conceitual necessária em *engine* de desenvolvimento de jogos com foco na Unity e na Construct. Na Seção 3.4, é apresentado o conceito de *Game Desing Document* e como este servirá para nortear as etapas seguintes desse trabalho.

3.1 Jogos

As definições para jogos existentes na literatura tem um espectro amplo e um caráter flexível quanto ao seu significado. Além disso, ao longo da história da humanidade os jogos sempre estiveram presentes e com isso seu significado se expandiu. "No sentido mais simples e educacional, uma interação em um relacionamento com propósitos de diversão e entretenimento, utilizando artifícios interpretativos ou ficcionais, vai de uma brincadeira a um jogo"(KISHIMOTO, 1994).

Jogos também podem ser definidos como "uma atividade voluntária exercida dentro de certos e determinados limites de tempo e espaço, seguindo regras livremente consentidas, mas absolutamente obrigatórias, dotado de um fim em si mesmo, acompanhado de um sentimento de tensão e alegria e de uma consciência de ser diferente de vida cotidiana"(HUIZINGA,).

Para os fins de compreensão deste trabalho será definido que um jogo "é qualquer atividade que necessita de pelo menos um jogador, possui um conjunto finito de regras e uma condição de vitória ou finalização"(ROGERS, 2014).

3.2 Jogos 2D

Jogos 2D são jogos que utilizam apenas duas dimensões, ou seja, não se é usado a dimensão da profundidade, apenas largura e comprimento. A ideia de jogo em duas dimensões surgiu com os primeiros vídeo games e foi utilizada fortemente nos de 60 a 90, pois o hardware de memória era mais limitado e jogos em duas dimensões geralmente consomem menos memória (KELLY, 2012).

Existem diversos jogos 2D que são muito famosos hoje em dia, pois utilizam uma abordagem com o foco de arte baseado em jogos clássicos e acabam chamando a atenção de

vários jogadores. Os jogos 2D clássicos são em boa parte jogos de plataforma que tem como característica uma tela estática, ou com rolagem, com a movimentação apenas de um personagem ao longo do plano cartesiano formado pela altura e largura da tela (KELLY, 2012).

Devido a simplicidade de implementação dessa categoria de jogos, muitos desenvolvedores independentes optam por desenvolver jogos 2D, pois a exigência de tempo para produzir os recursos é menor em relação a jogos 3D. Outra vantagem é a semelhança com desenhos feitos em papel real facilitando a criação, planejamento e execução (KHALIFA; SILVA; TOGELIUS, 2019).

Jogos 2D também estão presente em dispositivos móveis. Em geral, os processamentos nestas plataformas são menores se comparados com consoles e computadores, pois seu hardware não são construídos estritamente para a execução de games ou grandes taxas de processamento (KHALIFA; SILVA; TOGELIUS, 2019).

Com a evolução da tecnologia, a indústria de desenvolvimento de jogos tem crescido, tanto nas aplicações focadas em diversão, quanto em áreas relacionadas à educação, setores financeiros, assistencial e etc., se tornando uma indústria bastante competitiva. Porém, dada as características comuns a jogos da atualidade, a atividade de reuso de código funcional dos jogos se torna cada vez mais necessário e faz com que essas características iniciassem a ser ligadas em padrões, a ligação desses padrões resultou no surgimento das *engine* de desenvolvimentos de jogos.

3.3 *Engines* de desenvolvimento de jogos

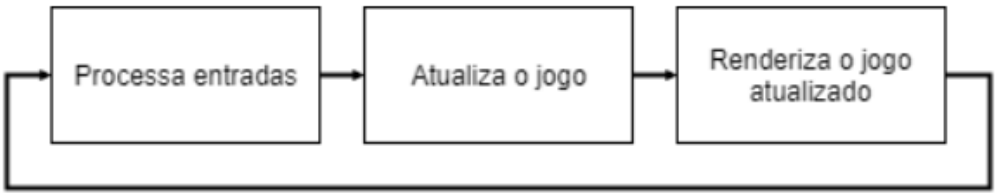
O desenvolvimento de jogos requer o conhecimento em várias áreas de atuação diferentes como efeitos sonoros, programação de algoritmos, desenhos gráficos, dentre outros. A construção de todas essas características de forma separada e desconjunta seria extremamente dispendiosa e iria requerer um enorme conhecimento técnico de várias áreas diferentes. Esse é o objetivo de *engine* de jogos, facilitar o desenvolvimento abstraindo dos desenvolvedores pontos muito específicos e aprofundados. Essa abstração varia de *engine* para *engine*. Esta facilidade faz com que desenvolvedores destinem seu esforço em aprender a usar da *engine* para conseguir desenvolver seus jogos utilizando todos os recursos necessários.

De forma mais objetiva, *engine* de jogos são ambientes de desenvolvimento, como uma IDE, com a responsabilidade de gerenciar o *loop* do jogo, que corresponde todas às fases do jogo, possuindo um conjunto de módulos e pode ser usada para criação de diferentes jogos

sem que seja necessário realizar grandes alterações. O *loop* de um jogo é responsável pela caracterização do processo fundamental do jogo, refletindo no seu ciclo de vida.

Ainda que geralmente esses *loops* de jogos executados pelas *engines* serem parcialmente complexos, todos seguem a mesma estrutura de execução. Essa estrutura passa pelo processamento das entradas do jogo, atualização dos estados do jogo, renderização do jogo em seu estado atual e na repetição desses processos (VALENTE; CONCI; FEIJÓ, 2005). Na Figura 1 é apresentada a estrutura do ciclo de vida de um jogo desenvolvido com o auxilio de uma *engine* de jogo.

Figura 1 – Ciclo de vida de uma *engine* de jogo.



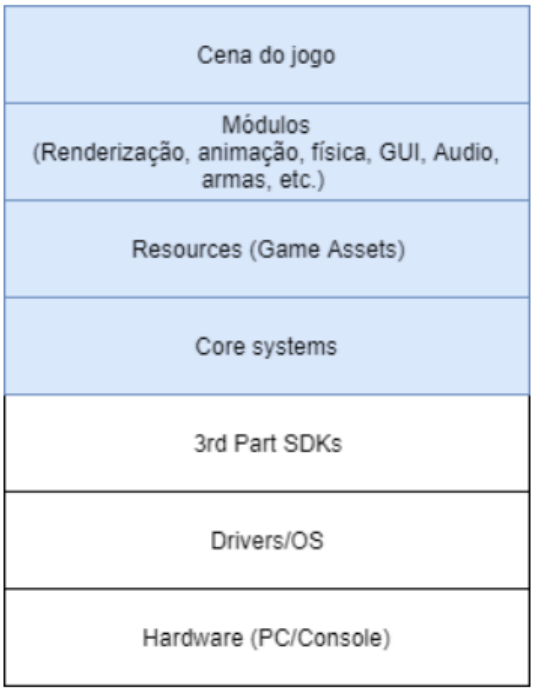
Fonte: Adaptado de (VALENTE; CONCI; FEIJÓ, 2005)

Além do ciclo de vida em geral das *engines* de jogos, a estrutura que as mesmas apresentam também mostra um certo padrão. Basicamente é uma relação necessária entre a cena do jogo e os recursos internos e externos à *engine* (GREGORY, 2017). De forma interna na *engine* existe o acesso aos módulos mais comuns de um jogo como a física dos objetos, a renderização dos gráficos, GUI, áudio, colisão, etc., além de contar com o *core system*, correspondente ao uso de memória e tratamento de erros da compilação do jogo em si. Outro recurso interno importante para uma *engine* de jogos é o *Game Assets*, que dá suporte a utilização de pacotes de imagens e sons para que possam compor as cenas do jogo (GREGORY, 2017).

Ainda com o auxilio da *engine*, alguns recursos externos que correspondem ao funcionamento do jogo no *smartphone*, ou, em um computador. Bibliotecas de processamento de gráficos também são utilizadas além dos *drivers* necessários para suportar toda execução funcional da aplicação (GREGORY, 2017). Na Figura 2, são mostrados todos os recursos estruturais presentes na maioria das *engines*. Os blocos em azul representam recursos fornecidos internamente pelas *engines* de jogos e em branco os recursos acessados externamente a elas.

Apesar da estrutura e do seu ciclo de vida semelhantes, atualmente as *engines* de jogos apresentam características distintas, em que a escolha específica de uma *engine* torna-se um motivo crucial no desenvolvimento de um projeto. Mesmo com o grande número de *engines* de jogos disponíveis, algumas se destacam por características que facilitam o desenvolvimento

Figura 2 – Estrutura de uma Game *engine*.



Fonte: (SILVA, 2018)

de um estilo de jogo, dentre elas estão Unity e a Construct 3, que serão apresentadas nas seções a seguir e evidenciada as características para o desenvolvimento de jogos 2D.

3.3.1 Unity

A *Unity* é uma game *engine* criada pela *Unity Technologies*. Sendo atualmente uma das *engine*s mais utilizadas no mundo pela eficiência e facilidade de desenvolvimento apresentada. Ela oferece aos desenvolvedores a possibilidade da criação de jogos 2D e 3D. O suporte dela contempla diversas APIs (Interface de Programação de Aplicações) como: Direct3D no Windows e Xbox 360, OpenGL no MacOS e Linux, OpenGL ES no Android e iOS, WebGL na Internet e também a novos dispositivos que utilizam realidade aumentada e realidade virtual. Anteriormente a parte de scripts utilizada pela plataforma era MonoDevelop que facilitava a utilização das linguagens C#, JavaScript ou Boo (que tem uma sintaxe de inspirada no Python). Em 2015, a Unity removeu o suporte a linguagem Boo, em 2017, a Unity anunciou que iria encerrar o suporte da linguagem JavaScript, que estava junto da *engine* desde sua criação (CALABRESE, 2014; GOLDSTONE, 2011). Atualmente o sistema de scripts da Unity tem o foco apenas em C#.

Em jogos 2D, a Unity permite a importação de *sprites*, que é um objeto gráfico bi ou tridimensional que se move numa tela sem deixar traços de sua passagem e um avançado renderizador de mundo 2D. Além disso, também são oferecidos aos desenvolvedores serviços como: Unity Ads, Unity Analytics, Unity, Unity Cloud, Unity Everyplay, Unity IAP, Unity Multiplayer, Unity Performance Reporting, Unity Collaborate e Unity Hub. Todos esses serviços ajudam os desenvolvedores entenderem melhor os usuários e conseguir monetizar e melhorar o trabalho realizado (CALABRESE, 2014; GOLDSTONE, 2011). As principais características da Unity são:

- Suporte para o uso de shaders;
- Programação em C# (principal) ou JavaScript (descontinuado);
- Suporte ao PhysX, incluindo detector de colisão, soft body e ragdoll;
- Compatibilidade com os navegadores (via o plugin Unity Web Player): Internet Explorer, Firefox, Safari, Opera, Google Chrome (versões anteriores a 45) e Camino;
- Compatibilidade com Blender, 3ds Max, Maya, Cinema 4D, Cheetah 3D, Softimage, modo, ZBrush, Lightwave, Photoshop, Fireworks, e "Substance".

As versões da Unity são lançadas anualmente. As contribuições nas atualizações contemplam a interface da IDE de desenvolvimento, assim como eficiência melhorada nos suportes que a *engine* dispõem. Existem diversos jogos famosos criados na plataforma que são premiados por aspectos de qualidade visual a jogabilidade e eficiência.

3.3.2 Construct 3

Para que cheguemos a *Construct* 3, a história da engine de desenvolvimento de jogos passa por algumas etapas. A primeira versão clássica, *Construct Clássica*, foi desenvolvida em outubro de 2007 por um grupo de estudantes, que disponibilizaram, inicialmente, a ferramenta de modo *open-source* usando *DirectX*. Um pouco depois, foi formada a empresa *Scirra*, que atualmente suporta toda a evolução da ferramenta. A época, essa versão definiu de forma abrangente o formato de uso com desenvolvimento visual dos jogos, ou seja, para desenvolver um jogo não seria necessário um entendimento amplo de programação em linha de código, usando *drag and drop*, uma técnica utilizada para incentivar a novos desenvolvedores entenderem a lógica de programação (SCIRRA, a). Outra opção foi, e ainda é, usar *scripts* em *Python* para o desenvolvimento dos jogos (BIGELOW, 2012).

Antes da descontinuação da *Construct Clássica*, foi lançada a *Construct* 2. Na época, era observada que seria necessário disponibilizar o acesso da *engine* em outros sistemas operacionais em busca de mais adoção da ferramenta, com isso, a principal mudança nessa versão foi no DirectX, tornando possível seu uso em novos sistemas operacionais e em *browser*. Além disso, a licença de uso não foi totalmente *open-source* como na primeira, apesar de ter uma versão grátis com alguns limites (SCIRRA, c). Uma diferença importante foi a retirada da execução de *scripts* em *Python* pois a ideia era ter a mesma ferramenta para todos os sistemas, incluindo no *browser*, e com a dificuldade de rodar *scripts Python* em um cliente web foi tomada uma decisão de projeto para a retirada da execução do mesmo. Esse ferramenta foi criada em 2011 e foi atualizada até 2019, porém em julho de 2020 as licenças de uso não foram mais vendidas para que todo o foco da equipe que desenvolve a *engine* ficasse na ferramenta atual da empresa, que é *Construct* 3, usada no presente trabalho. Em 2015 a equipe de desenvolvimento já sabia do rumo que a *engine* deveria tomar, e relatou para comunidade todo esse processo de atualização que a ferramenta passaria, já anunciando a *engine* atual *Construct* 3 (SCIRRA, b).

A *Construct 3*, por sua vez, é uma engine que se baseia em HTML5, para construir jogos 2D. Seu lançamento com novidade de ser usada sistemas *Mac* e *Linux* e com suporte a mais linguagens de programação. Porém, a principal diferença é um *SDK* oficial para *Android* suportando *plugins* de terceiros para expansão dos jogos desenvolvidos (SCIRRA, b). Desde seu lançamento em 2017, foram feitas algumas atualizações no plano de pagamento para o uso da ferramenta como novos recursos de programação utilizando a *engine*.

Apesar de inicialmente a proposta era ser uma *engine* lançada para desenvolver jogos no *browser* com *DirectX*, atualmente, com as atualizações, vários suportes são fornecidos. Dentre esses estão:

- Windows, MacOS e Linux usando NW.js;
- Android e iOS usando Cordova;
- Windows Store usando UWP (que possibilida uma integração com consoles mais modernos *Xbox*).

Com a popularização da ferramenta, apesar de ser paga, vários jogos são lançados utilizando-a. Existem diversos jogos independentes feitos na *Construct 3* presentes nas plataformas que agregam jogo como a *Steam* prestando suporte a diversos sistemas.

3.4 *Game Design Document*

Para possibilitar a comparação das ferramentas descritas anteriormente, é necessário o desenvolvimento de um jogo. Com isso, apresenta-se o conceito de *Game Design Document*, ou, GDD, que de forma prática é um documento de *software* que descreve diversas características consideraveis para o desenvolvimento do jogo (OXLAND, 2004). De maneira geral, os artefatos necessários do GDD descrevem o ciclo de vida do jogo, a história com o contexto, a estrutura prática para que o time de desenvolvimento tenha conhecimento de propriedades essenciais, e, o conceito do jogo explicando a ideia principal dele (III, 2004).

Apesar de não existir uma estrutura rígida que defina o que de fato é um GDD, deve haver uma estrutura base para possibilitar o desenvolvimento do jogo. Com isso temos os seguintes tem tópicos que são apresentados a seguir:

- História;
- Personagens;
- Levels;
- Jogabilidade;
- Arte;
- Som;
- Mecânicas do jogo.

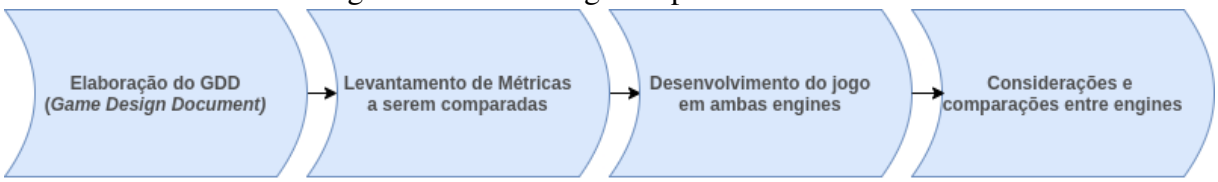
Esses pontos são essenciais para que, tanto a equipe de desenvolvimento quanto a produtora consigam executar suas atividades. A equipe de desenvolvimento deve conseguir ter *features* técnicas como um norte na etapa de desenvolvimento e o time de pós-produção/venda deve conseguir classificar o jogo, e principalmente, ter uma história para contar ao público alvo.

Neste trabalho, o GDD é levantado e apresentado para que possibilite o desenvolvimento de um jogo porém, a finalidade é somente acadêmica, ou seja, para que o jogo seja desenvolvido e de fato tenha a comparação das duas *engines*, o GDD serve para nortear toda a etapa de desenvolvimento.

4 PROCEDIMENTOS METODOLÓGICOS

Neste capítulo, são descritas todas as fases impostas para a execução do presente trabalho. Na Figura 3, é mostrado o fluxo de trabalho que foi empregado. Inicialmente é apresentado o GDD do jogo desenvolvido, para que seja estabelecido o cenário de desenvolvimento em que cada *engine* foi submetida. Com isso, serão apresentadas as métricas de comparação que levam em conta aspectos práticos do desenvolvimento de um jogo 2D. Após o desenvolvimento do jogo em ambas as plataformas, serão aplicados as métricas de avaliação a fim de comparar a fase de desenvolvimento e o desempenho do jogo em ambas *engines*.

Figura 3 – Metodologia do presente trabalho.



Fonte: Elaborado pelo autor (2021).

4.1 Desenvolvimento do *Game Desing Document*

Antes mesmo da criação do GDD do jogo, foram feitas investigações em portais de jogos independentes para servir de inspiração para a temática do jogo a ser desenvolvido. Com isso realizado, foi-se desenvolvido o GDD do jogo 2D a ser desenvolvido em ambas as *engines*.

A ideia geral do jogo serve para contemplar aspectos comparáveis entre as *engines* e também simplificar o desenvolvimento, contando com aspectos comuns a jogos 2D. A falta de robustez foi pensada para acelerar o processo de desenvolvimento e aprendizagem em ambas as *engines*. O GDD foi criado com tópicos importantes para o entendimento do jogo que foi desenvolvido, como não é definido um padrão exemplar pro GDD, os aspectos mais importantes pro desenvolvedor foi levantado.

4.2 Métricas de avaliação

A fim de avaliar e comparar as *engines* de desenvolvimento de jogos citados nas sessões anteriores, foram utilizadas métricas que evidenciam as diferenças entre processos relevantes no desenvolvimento de jogos 2D.

As métricas dispostas tem base teórica no ciclo de desenvolvimento de uma *engine* assim como nos trabalhos relacionados (ŠMÍD, ; SILVA, 2018; CAVALCANTE; PEREIRA, 2018). As métricas são:

- *TilesMap*: *Tiles* são cada bloco repetitivos que constituirão o piso e paredes de cada nível do jogo. Este será o primeiro ponto analisado e implementado devido a grande importância em relação ao movimento do jogador, dos inimigos e mecânica de luz. No caso deste trabalho estaremos usando 2 *TilesMap* 1 para o piso, e 1 para as paredes;
- Sistema de câmera: Sistema de câmera é a visão apresentado para o jogador no momento atual, embora outras coisas estejam acontecendo em outras partes do jogo. Neste jogo o sistema de camera sempre segue o jogador, mantendo-o sempre no centro;
- Movimentação do jogador: O movimento do jogador é a mudança de posição do avatar do jogador no *tiles* recentemente criados. O movimento é feito nas 4 possíveis posições: cima, baixo, direita e esquerda. A ação pode ser realizada no piso normal e não nos tiles de parede;
- Sistema de luz: Sistema de luz simula a luz real, em que uma fonte luz clareia uma determinada área. Neste jogo não há iluminação natural e é gerada por uma fonte de luz que o jogador possui.
- Sistema de movimento de IA: o movimento dos inimigos até o jogador de maneira automática;
- Som: sistema de som, basicamente a processo da aplicação de tocar algum som quando determinada ação acontece;
- Transição de levels: O sistema de levels representa a transição de uma fase do jogo para outra.

A escolha de métricas estão diretamente relacionadas com aspectos de desenvolvimento do jogo, ou seja, para cada ponto será observado e descrito como resultado os passos e as dificuldades encontradas em cada *engine*.

4.3 Desenvolvimento do jogo

Com o GDD e as métricas de comparação definidos, iniciou-se o processo de desenvolvimento do jogo em cada *engine*. Durante essa fase, cada *engine* apresentou prós e contras em relação a diversos aspectos voltadas para o desenvolvedor, desde a facilidade de usar a ferramenta até suporte da própria comunidade para consulta quando havia algum empecilho.

Isso não necessariamente torna uma melhor que a outra, porém, a velocidade de desenvolvimento muda e também o poder de criação de detalhes.

Alguns elementos podem ser considerados na hora de comparar o processo de desenvolvimento em si utilizando cada ferramenta. Como o foco desse trabalho visa principalmente a comparação de aspectos técnicos, será apresentado de forma breve algumas considerações feitas durante o processo de desenvolvimento em cada *engine*, levando em conta aspectos mais brandos como a curva de aprendizagem, licenciamento e custos, documentação e suporte da comunidade.

5 GDD: LUZ NO FIM DO TÚNEL

Neste capítulo, será exposto o *Game Desing* do jogo que foi desenvolvido visando a comparação das *engines*.

5.1 Descrição Geral do Projeto

Este jogo se encaixa no modelo de puzzle 2D *top-down*, em que o jogador se encontra em um labirinto com monstros e armadilhas. O jogador não possui armas, nem meios de combater os inimigos diretamente, assim terá que usar estratégia e planejamento para vencer as fases e avançar até conseguir alcançar o a luz no fim do túnel.

Os túneis possuem vários monstros e armadilhas que eliminam o jogador ao fazer contato. As armadilhas também derrotam os monstros. O jogador possui uma lanterna que pode ser ligada e desligada, ele a usará para iluminar o caminho e decidir para onde se movimentar. Esses movimentos podem ser feitos tanto no claro quanto no escuro. Com a luz de sua lanterna, ele poderá ver se o caminho possui monstros, armadilhas ou se está seguro.

Ao iluminar um monstro o mesmo irá perseguir o jogador enquanto ele estiver com a lanterna ligada e parando quando desligada. Assim, a mecânica para vencer os inimigos será identificar onde estam as armadilhas e atrair o monstros para elas, acendendo e apagando a luz quando necessário.

1. História

   O jogo de passa em um contexto pós-apocalíptico com zumbis em 2030 em que o único sobrevivente de um grupo entra em um túnel subterrâneo fugindo de uma legião de zumbis. O mesmo túnel está cheio de zumbis e a única maneira de sair será usando suas habilidades para escapar e evitar-los, achando assim a luz no fim do túnel.

2. Gameplay
   - O jogador entrou no túnel fugindo de uma legião gigante, assim seu objetivo é achar a saída do túnel;
   - Os tuneis são escuros e estão cheios de zumbis e armadilhas;
   - O jogador move uma posição por vez;
   - O jogador possui uma lanterna que mostra apenas uma posição a frente e pode ser ligada e desligada;
   - O jogador morrerá ao realizar contato direto com um zumbi;

- Jogador ou zumbi morrerá ao cair em uma armadilha;
- Apos um zumbir avistar o jogador o mesmo será atraído pela luz. Assim, a única maneira de matar zumbis é atraindo-os pra armadilhas presentes nos tuneis;
- A cada movimento realizado pelo jogador, o mesmo consome energia e então deve procurar suprimentos pelos tuneis.

3. Personagens

   Jason, o personagem principal é possui 25 anos e o único sobrevivente de grupo que foi atacado por uma grande legião de zumbis. Infelizmente ao fugir ele só pode carregar sua lanterna e terá que usa-la de maneira criativa para lhe dar com as ameaças dos tuneis. Ele possui a habilidade de andar, ligar, desligar sua lanterna e consumir suprimentos.

4. Controles
   - O jogador controla usando o teclado;
   - Seta para esquerda – mover para esquerda;
   - Seta para cima – mover para cima;
   - Seta para direita – mover para direita;
   - Seta para baixo – mover para baixo;
   - Espaço – Ligar/desligar luz.

5. Câmera

   A câmera e o movimento do personagem são no estilo *Top-Down*, em que o jogador verá o seu personagem de cima para baixo, assim poderá movimenta-lo nas 4 possíveis direções.

6. Universo do Jogo

   O jogo possui uma atmosfera de terror, em geral causada pela visão limitada do jogador e o possível encontro com uma ameaça cujo o mesmo não possui formas de combater diretamente. O jogo todo se passa dentro dos tuneis, cada fase representa uma parte do túnel que possui seu fim no ultimo deles.

7. Inimigos
   - Zumbis
     - Matam o jogador ao fazerem contato direto;
     - São atraídos apenas pela luz;
     - Morrem ao encostarem em armadilhas.
   - Armadilhas
     - Matam o jogador ao fazerem contato direto.

8. Plataformas

   Desktop.

9. Escopo do projeto

   Custo prazo:

   - Todos os *assets* e *engines* foram gratuitos, logo não haverá custos monetários;
   - O jogo foi produzido no período entre 20 de janeiro de 2021 e terminou em 28 de fevereiro de 2021.

   Equipe:

   - Programador
     - Gelderson Bezerra Alves
   - Responsabilidades
     - Programador e desenvolvedor geral
     - Designer Geral

10. Ativos Necessários
    - Figuras 2D
      - Jogador
      - Lanterna
      - Inimigo
      - Chão
      - Parede
      - Armadilha
      - Passagem para próximo túnel
    - Sons
      - Som do ambiente do túnel
      - Som dos passos 1
      - Som dos passos 2
      - Som de grito
    - Código
      - Scripts de Personagens
        * Movimentar
        * Ligar/desligar luz
      - Scripts de Inimigos

        - Movimentar
        - Perseguir a luz
- Animações de personagens
    - Jogador
        - Movimento
    - Inimigo
        - Movimento

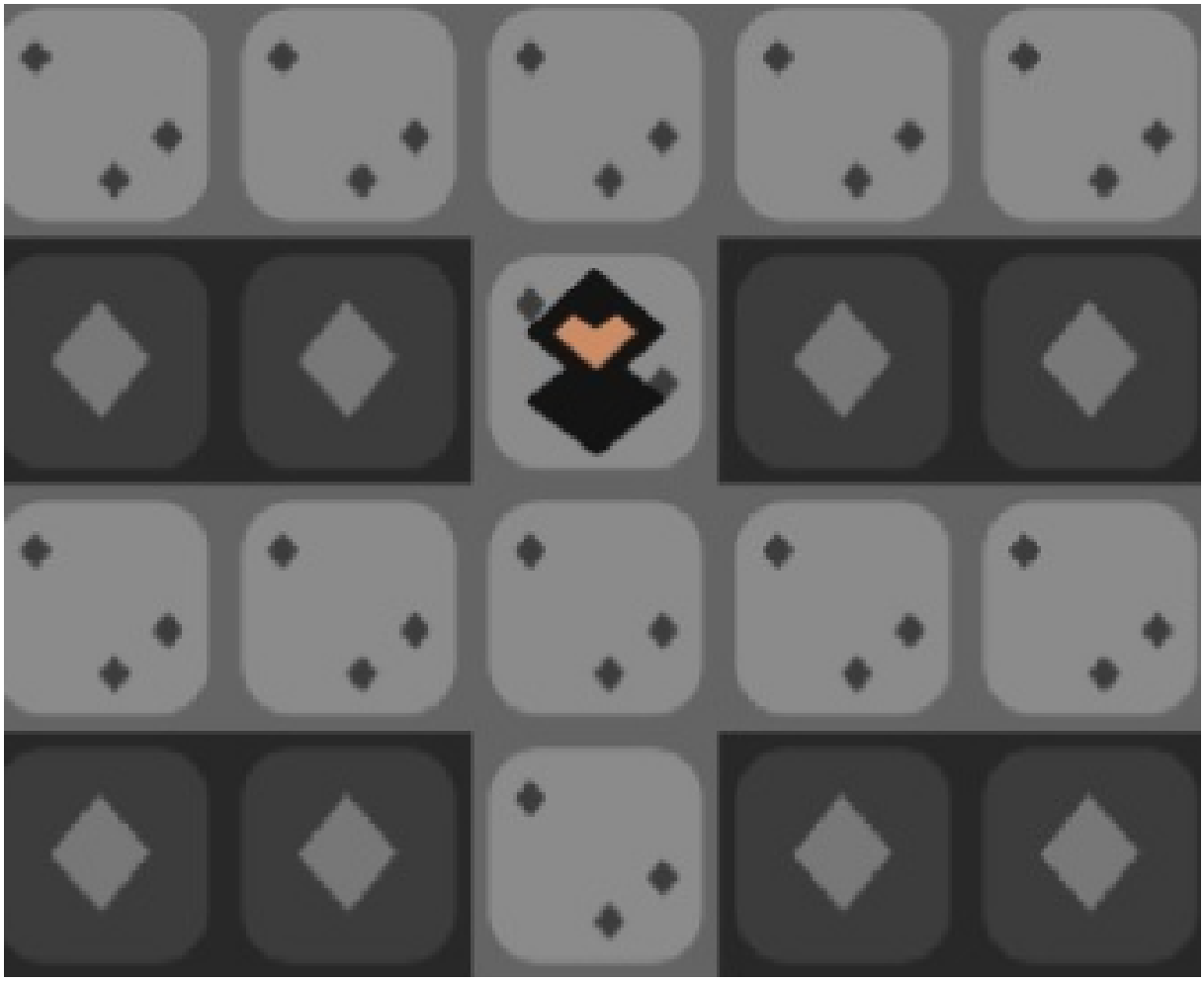
6 RESULTADOS

Neste capítulo, são apresentados os resultados obtidos durante o trabalho. É importante enfatizar que o principal objetivo do trabalho é comparar aspectos técnicos bem definidos no processo de desenvolvimento de um jogo 2D entre as *engines Unity* e *Construct* 3.

6.1 Luz no fim do túnel

Na figura 4, é apresentada uma tela do jogo desenvolvido. Como as mesmas imagens foram utilizadas tanto na *Unity* quanto na *Construct* 3, o jogo tem a mesma tela em ambas. Além disso, a figura 5 tem todas as *sprites* usadas individualmente para cada elemento presente no jogo.

Figura 4 – Personagem no labirinto



Fonte: elaborado pelo autor

Figura 5 – *Sprites* do jogo

Fonte: elaborado pelo autor

Como o objetivo do trabalho é sintetizar os resultados utilizando as métricas que levam em consideração aspectos técnicos no desenvolvimento de um jogo 2D, o *design* das *sprites* foi simples para acelerar o processo de desenvolvimento. A jogabilidade simples também tem a mesma motivação. Como foi visto nas imagens acima, de fato, todos os ativos do jogo foram desenvolvidos.

6.2 Etapas de desenvolvimento

Para avaliar a fase de desenvolvimento do jogo são utilizadas as parâmetros de comparação, que levam em conta aspectos técnicos da fase de desenvolvimento, definidas na metodologia. As próximas subseções especificam de forma prática o passo a passo dos procedimentos feitos em cada uma das *engines*, para cada métrica definida.

6.2.1 TilesMap

- *Construct* 3
  - Criação de *TilesMap*:
    1. Selecione o Layout do game Tile Width e Tile Height para definir o tamanho dos tiles, neste caso 64 x 64 ;
    2. Click no lado direito da tela, insert new object, TileMap, para criar o Tile Chão;
    3. Selecione o arquivo a ser usado e ajuste o tamanho de de altura e largura para o valor dos tiles do layout, 64x64, neste caso;
    4. Ajuste o size do tile para o tamanho do layout, assim toda a tela será coberta;
    5. Repita o processo para o Tile de parede.
  - Utilzando os Tiles:
    1. Selecione o Layout e marque a opção show grid para garantir a facil visualização do mapa para o uso de Tiles;

2. Selecione o TileMap no layout e depois o tile especifico a ser usado na aba TileMap do projeto, neste caso o de piso;
3. Selecione o botão de lápis e depois basta clicar nos blocos da layout para criar o piso no mapa;
4. Repita o processo para o tile da parede.

– Deixando a parede sólida:

1. Selecione no TileMap parede e no menu de properties clique em Edit Behaviors depois Solid e marque a opção enable, assim o player não conseguirá atravessar as paredes.

• *Unity*

– Criação de *TilesMap*:

1. Clique com botão direito na Hierarquia depois 2D object e TileMap, o objeto Grid será criado contendo 1 tilemap;
2. Duplique o tileMap para existir os níveis de Tiles necessários, piso e paredes;
3. Selecione o segundo tileMap e no Additional Settings, mude o valor de Order in Layer para 1, assim este Tile ficará acima do anterior;
4. Clique em Window > 2D > TilePallete, para abrir a aba para edição de tiles;
5. Arraste os Sprites de Piso e parede para o aba, para poderem ser usados nos TilesMaps.

– Utilzando os Tiles:

1. Selecione o primeiro tileMap, tile de piso no Tile TilePallete e crie a parte inferior do level;
2. Selecione o segundo tileMap, tile de parede no Tile TilePallete e crie a parte superior do level;

– Deixando a parede sólida:

1. Selecione o objeto tilemap parede;
2. Adicione os seguintes componentes: Rigidbody2d e selecione o Kinematic na Opção body type TileMap Collider 2D e marque a opção used by composite Composite Collider 2d.

Nesta métrica, tanto a complexidade para organizar o sistema de tiles quanto a facilidade na sua utilização se mostrou bem similar em ambas as *engines*.

6.2.2 Sistema de Câmera

- *Construct* 3
    1. Crie um novo Sprite vazio, chamado câmera;
    2. Na aba esquerda de Behaviors adicione o ScrollTo e marque a opção Enable;
    3. Na aba Common ajuste o tamanho para 250x250;
    4. Ajuste o tamanho do tile para o tamanho do layout, assim toda a tela será coberta;
    5. Selecione a aba superior Event Sheet e no Event Every Tick crie uma nova ação e selecione as seguintes ações:
        - Câmera;
        - Ajustar o tamanho da camera;
        - Ajuste X para lerp(Self.X, Player.X, 0.04);
        - Ajuste Y para lerp(Self.Y, Player.Y, 0.04)
- *Unity*
    1. Selecione o objeto Main Camera e arraste para o Objeto player, assim o camera seguirá o player;
    2. Ajustar o tamanho da camera.

A *Unity* foi bem simples e eficiente, assim como pode ser observado pela diferença e complexidade nos passos descritos. Também foi notado que há um suporte maior em relação a manipulação dos comportamentos de câmera, apesar de tornar mais complexo, na *Unity*.

6.2.3 Movimento do Player

- *Construct* 3
    1. Selecione o player e na sessão Behaviors adicone o TileMoviment;
    2. Mude os seguintes parametros de TileMoviment:
        - Grid width para 64;
        - Grid hight para 64.
- *Unity*
    1. Crie o objeto player arrastando ele para a dentro da câmera e ajustando a escala, deixando-o menor que os tile;
    2. Selecione o objeto player e no Inspector clique em Add component, adicione um Rigidbody 2D e um Box collider 2D;

3. Adicione o script de PlayerMoviment, para criar o movimento do player. Este script detecta as teclas pressionadas e aciona o movimento do jogador.

A construção do sistema de movimentação do player pôde ser feita de forma bastante simples em ambas*engines*, porém, na desenvolver esse passo na *Construct* foi mais simples, pois de forma nativa já presta suporte a vários estilos de movimento. A *Unity*, embora eficaz, é quase que completamente dependente do código, o que aumenta sua complexidade neste ponto.

6.2.4 Sistema de luz

- *Construct* 3
    1. No menu de layers clique em add layer on top para criar o novo layer que será a escuridão;
    2. Nas Propriedades do layer desmarque a opção transparente, mude o Background color para preto e Force own texture para yes;
    3. Crie um novo sprite desenhe um circulo com o pincel, que funcionará como uma fonte de luz, na aba Effects selecione Destination out na opção Blend mode;
    4. Edite o sprite novamente e na opção superior resize ajuste os valores Height e Width para 88.
- *Unity*
    1. Na opção superior Window escolha Redering e depois Lighting Settings;
    2. Em Environment Lighting, ajuste Ambient Color para preto;
    3. Na aba de objetos, clique com o botão direito do mouse e selecione Spot Light. Ajuste os seguintes parametros no Inspector:
        - Position/Z: -0.72
        - Rotation/X: 0
        - Range: 2.55
        - SpotAngle: 97.9
        - Intensity: 2.51

Nesta métrica, a *Construct* se destacou levemente pela simplicidade de desenvolvimento. Além disso, a criação desse sistema não apresentou impacto de performance. Já a *Unity* apresenta mais opções de configuração para as luzes e um método mais realista, porém, foram notados impactos de performance, o que pode ser um problema para um jogo de caráter independente, além de tornar a implementação em geral mais complexa.

6.2.5 Sistema de movimento de AI

- *Construct* 3
  1. Criar um novo sprite monstro e adicionar o sprite do monstro;
  2. Adicionar o Behavior TileMoviment e mudar os seguintes paramentros:
     - Grid Width: 64
     - Grid Height: 64
     - Grid off set X: 32
     - Grid off set Y: 32
  3. Adicionar o Behavior Pathfinding e mudar os seguintes paramentros:
     - Cells size: 64
     - Cell border: -1
     - Obstacles: Solids
  4. Criar os seguintes eventos e subeventos: AIMovimentConstruct.png
- *Unity*
  1. Baixar os arquivo da biblioteca do repositório[1] e adicionar no projeto.
  2. Criar um novo EmptyObject, renomie para NavMesh;
  3. Adicione o componente NavMeshSurface2D;
  4. Selecione os 2 tiles, parade e piso, dentro do obejto grid e adicione o componente NavMeshModifier;
  5. Selecione o tile Piso, na sessão do NavMeshModifier marque Override Area, e no Area Type Selecione Walkable;
  6. Repita o procesos acima para o Tile Parede mas selecione Not Walkable;
  7. Selecione o objeto NavMesh, mude o valor Rotation X para -90, na opção Agent Type selecione Open Agent Setting e mude os seguintes valores: Radius 0.2 e Height 0.5
  8. Na ultima sessão, Nav Mesh Data selecione Bake, e será gerado o caminho em que os inimigos poderam se movimentar e seguir o player;
  9. Crie um novo sprite para ser o monstro;
  10. Adicionar o seguinte script para ativar o movimento do inimigo: UnityAiMovimentScript.png. Assim o inimigo deve seguir o player nos caminhos marcados como Walkable.

[1] https://github.com/h8man/NavMeshPlus

Esse foi o requisito mais complexo em ambas as *engines*. A *engine* Construct fornece o Behavior Pathfinding que gera um conjunto de células com o caminho mais rápido pra um ponto. Baseado nisso comparo a diferença de posição do monstro em relação a próxima célula, simulo o respectivo movimento para este ponto e depois gero um novo caminho. Em geral na *Unity* a construção de caminhos mais curtos entre 2 pontos é feito de forma bem complexa, com alguns **scripts** de computação de nodes. Diante dessa complexidade foi usada uma biblioteca para facilitar o processo em 2D.

6.2.6 Som

- *Construct* 3
    1. Clicar na pasta Sounds do projeto, Import sounds e escolher os arquivos de audio;
    2. Adicionar a action Audio no Event Sheet, no evento onde o audio deve tocar com o volume igual a 5.
- *Unity*
    1. Crie um objeto AudioSource;
    2. Adicione o arquivo de audio no projeto, e no parametro AudioClip do AudioSource;
    3. No script do player crie uma variável AudioSource que recebera o objeto AudioSource recem criado;
    4. Chame o .Play(); da variável para tocar o audio designado uma vez.

O sistema de som foi bem simples de ser implementado e usado em ambas as *engines*, mas a *Unity* possui mais opções de variação e adaptação de som, somando isso a facilidade da implementação, faz ela ser superior nesta métrica.

6.2.7 Transição de levels

- *Construct* 3
    1. Crie um novo Layout, clicando na pasta Layouts e depois Add Layout;
    2. Adicione a action System Go to layout e especifique o nome do layout.
- *Unity*
    1. Crie uma nova Scene, clicando com o lado direito na pasta scenes, na aba projeto;
    2. Adicione a nova Scene ao build do projeto, indo em File, Build Settings, Add Open Scenes, e marca a scene recem criada;
    3. Use esse código para fazer a transição Application.LoadLevel([numero da scene]);.

Neste caso, ambas as *engines* possuem interface que permite uma fácil implementação, tornando a transição de level do player bem fluida.

6.3 Avaliação de desenvolvimento

O desenvolvimento na *Construct* 3, de forma geral, se mostrou bem simples durante todo o processo. Tudo que foi necessário para criação de ativos do jogo já estavam presentes na ferramenta. A partir da experiência prévia do desenvolvedor, o uso da ferramenta se tornou bem intuitivo e direto, sem sinuosidade. A comunidade revelou-se ativa, apesar de não ser tão forte como a da *Unity*, viu-se muitos *posts* antigos e novos. Como o jogo proposto é, relativamente, simples, tudo fluiu de forma rápida durante o processo de desenvolvimento. A técnica de *drag and drop* ficou bem evidente durante essa fase, com adição de *scripts* em *JavaScript* somente ao fim de um fluxo bem definido. O repositório[2] do jogo desenvolvido utilizando *Construct* 3 pode ser encontrado em https://github.com/Gelderson/Luz-no-fim-do-tunel-CONSTRUCT-3.

A *Unity* se mostrou uma ferramenta mais completa que a *Construct* 3. Com isso, o processo de desenvolvimento, naturalmente, tornou-se mais complexa, mas também, possibilitando pro desenvolvedor mais opções de desenvolvimento. O uso de *scripts*, utilizando *C#*, foi bem mais presente nessa ferramenta, apesar de também ter sido utilizado a técnica de *drag and drop*.

Porém, apesar dessa complexidade, a comunidade em volta da *Unity* é bem maior, apresentando mais alternativas de conteúdo para resolução de problemas durante a fase de desenvolvimento. Isso só reforça que de fato, ela é a maior *engine* para o desenvolvimento de jogos. O repositório[3] do jogo desenvolvido utilizando *Unity* pode ser encontrado em https://github.com/Gelderson/Luz-no-fim-do-tunel_UNITY.

Uma diferença importante entre as *engines* é a construção do executável do projeto. Enquanto a disponibilização utilizando a *Unity* é gratuita, já integrada no software, a *Construct* 3 obriga você pagar a licença anual para gerar o executável do jogo. Portanto, se o objetivo do desenvolvedor for disponibilizar o jogo em alguma plataforma que agrega jogos independentes, essa questão deve ser levada em consideração. Cada *engine* tem seus pontos positivos e negativos, então, esse trabalho evidencia as principais diferenças entre o uso de ambas as ferramentas para que, quem tiver acesso, consiga entender o cerne da questão e faça a escolha mais adequada.

---

2 https://github.com/Gelderson/Luz-no-fim-do-tunel-CONSTRUCT-3
3 https://github.com/Gelderson/Luz-no-fim-do-tunel_UNITY

7 CONSIDERAÇÕES FINAIS

Com a evolução do acesso a tecnologia, uma das vertentes que maior ganhou demanda foi a de entretenimento. Dentro disso, os jogos aparecem como uma opção muito consumido, desde crianças até adultos. Existem diversos tipos de jogos diferentes contemplando inúmeros perfis. Por óbvio, as ferramentas e as formas de desenvolver jogos modernos também evoluíram junto com esse mercado.

O presente trabalho teve como objetivo utilizar duas das ferramentas modernas de desenvolvimento de jogos 2D, através da criação de um jogo que contemplasse aspectos técnicos a serem comparados entre as ferramentas. Essas ferramentas foram escolhidas baseado no critério de popularidade de desenvolvimento de jogos independentes. As ferramentas apresentaram semelhanças e diferenças no processo de desenvolvimento que foram apontadas ao decorrer deste trabalho.

Parte dos trabalhos acadêmicos semelhantes a este, focam em métricas utilizadas pós ou pré desenvolvimento do jogo propriamente dito. Isso é a principal diferença entre estes e o atual trabalho, que foca principalmente em métricas considerando aspectos técnicos medíveis durante a fase de desenvolvimento do jogo.

As *engines* escolhidas apresentaram uma curva de aprendizagem rápida para quem já tem uma experiência prévia em desenvolvimento de *software*. Em um cenário que se deseja desenvolver um jogo com mais detalhes e mais controle sobre aspectos técnicos do jogo, ficou evidente nesse trabalho que a *engine* a ser escolhida é a *Unity*. Em contrapartida, para cunho educacional, a *Construct* 3 se destaca, pois o grau de desenvolvimento do jogo passa apenas pela capacidade de usar a ferramenta a nível de *drag and drop*.

O jogo desenvolvido que possibilitou essa conclusão e comparação apresenta um cenário simples de jogabilidade, principalmente para que houvesse tempo viável de desenvolvimento em ambas as *engines*. Para um trabalho futuro, entende-se que aumentar a complexidade desse jogo para conseguir levantar métricas mais complexas é uma possibilidade. Provavelmente, um jogo mais complexo evidencie de mais ainda as facilidades e liberdades que o desenvolvedor pode ter com cada *engines*. Além disso, o uso de outras *engines* também pode ser considerado.

REFERÊNCIAS

BIGELOW, D. **Construct Game Development Beginner's Guide**. [S.l.]: Packt Publishing Ltd, 2012.

CALABRESE, D. **Unity 2D game development**. [S.l.]: Packt Publishing Ltd, 2014.

CAVALCANTE, C. H. L.; PEREIRA, M. L. A. **Comparativo entre Game Engines como Etapa Inicial para o Desenvolvimento de um Jogo de Educação Financeira**. [S.l: S.n], 2018.

FERNANDES, A. M.; CAMARA, B. H. P.; PASCHOAL, A. R.; DAMASCENO, E. F. Inteligência artificial aplicada a jogos de tabuleiro com realidade aumentada. In: **15º Workshop de Realidade Virtual Aumentada, Marília-SP**. [S.l.: s.n.], 2015. v. 1.

FERNANDES, D. R. Jogos digitais e história: Uma experiência interdisciplinar e multidisciplinar. **Redin-Revista Educacional Interdisciplinar**, [S.l], v. 7, n. 1, 2018.

GAMEFROMSCRATCH.COM, C. **GameMaker Studio 2 Released**. [S.l], 2017. Disponível em: https://www.gamefromscratch.com/post/2017/03/09/GameMaker-Studio-2-Released.aspx. Acesso em: 24 ago.2019.

GOLDSTONE, W. **Unity 3. x game development essentials**. [S.l.]: Packt Publishing Ltd, 2011.

GREGORY, J. **Game engine architecture**. [S.l.]: AK Peters/CRC Press, 2017.

HUIZINGA, J. **Homo Ludens**: o jogo como elemento da cultura, 1938. Tradução de JP, [S.l: s.n], 2008.

III, R. R. **Game design:** theory and practice. [S.l.]: Jones & Bartlett Publishers, 2004.

KELLY, C. **Programming 2D games**. [S.l.]: CRC press, 2012.

KHALIFA, A.; SILVA, F. de M.; TOGELIUS, J. Level design patterns in 2d games. In: IEEE. **2019 IEEE Conference on Games (CoG)**. [S.l.], 2019. p. 1–8.

KISHIMOTO, T. M. O jogo e a educação infantil. [S.l]: **Perspectiva**, v. 12, n. 22, p. 105–128, 1994.

OXLAND, K. **Gameplay and design**. [S.l.]: Pearson Education, 2004.

O'HAGAN, A. O.; COLEMAN, G.; O'CONNOR, R. V. Software development processes for games: A systematic literature review. In: SPRINGER. **European Conference on Software Process Improvement**. [S.l.], 2014. p. 182–193.

ROGERS, S. **Level Up! The guide to great video game design**. [S.l.]: John Wiley & Sons, 2014.

SCIRRA, C. **Construct Classic**. [S.l], 2011. Disponível em: https://www.scirra.com/construct-classic. Acesso em: 01 mar. 2021.

SCIRRA, C. **The future of construct**. [S.l], 2015. Disponível em: https://www.construct.net/en/blogs/construct-official-blog-1/future-construct-836 . Acesso em: 02 mar. 2021.

SCIRRA, C. **How construct 2 licenses work**. [S.l], 2011. Disponível em: https://www.construct.net/en/tutorials/how-construct-2-licenses-work-64 . Acesso em: 02 mar. 2021.

SILVA, D. L. da. **Análise de ferramentas e desenvolvimento de jogo para treinamento de paratletas**. [S.l: s.n], 2018.

SILVA, R. E. da; MARTINS, S. W. Ensino de ciência da computação através do desenvolvimento de jogos. In: **VII Congresso Iberoamericano de Informática Educativa**. [S.l.: s.n.], 2015. p. 1286–1295.

ŠMÍD, A. **Comparison of Unity and Unreal Engine**. Bachelor Thesis. Praga, República Checa: DCGI/Faculty of Electrical Engineering, 2017.

UNITY. **Unity**. [S.l]: 2019. Disponível em: https://unity3d.com/pt/unity/features/multiplatform . Acesso em: 15 ago. 2019.

VALENTE, L.; CONCI, A.; FEIJÓ, B. Real time game loop models for single-player computer games. In: **Proceedings of the IV Brazilian Symposium on Computer Games and Digital Entertainment**. [S.l.: s.n.], 2005. v. 89, p. 99.