CENTRO UNIVERSITÁRIO CHRISTUS

SISTEMAS DE INFORMAÇÃO

DAVID EINSTAIN JUDA CARNEIRO

OLD BONES: UM JOGO DE AÇÃO INTERATIVO DE TIRO RETRÔ DESENVOLVIDO EM UNITY

FORTALEZA

2023

DAVID EINSTAIN JUDA CARNEIRO

OLD BONES: UM JOGO DE AÇÃO INTERATIVO DE TIRO RETRÔ DESENVOLVIDO EM UNITY

Trabalho de Conclusão de Curso (TCC) apresentado ao curso de Sistemas de Informação do Centro Universitário Christus, como requisito parcial para obtenção do grau de bacharel em Sistemas de Informação.

Orientador: Prof. Dr. Gleidson Sobreira Leite

FORTALEZA

2023

Dados Internacionais de Catalogação na Publicação
Centro Universitário Christus - Unichristus
Gerada automaticamente pelo Sistema de Elaboração de Ficha Catalográfica do
Centro Universitário Christus - Unichristus, com dados fornecidos pelo(a) autor(a)

C289o Carneiro, David Einstain Juda.
Old bones: um jogo de ação interativo de tiro retrô desenvolvido em Unity / David Einstain Juda Carneiro. - 2023.
40 f. : il. color.

Trabalho de Conclusão de Curso (Graduação) - Centro Universitário Christus - Unichristus, Curso de Sistemas de Informação, Fortaleza, 2023.
Orientação: Prof. Dr. Gleidson Sobreira Leite.

1. Desenvolvimento. 2. Jogos digitais. 3. Unity. I. Título.

CDD 004.07

DAVID EINSTAIN JUDA CARNEIRO

OLD BONES: UM JOGO DE AÇÃO INTERATIVO DE TIRO RETRÔ DESENVOLVIDO EM UNITY

Trabalho de Conclusão de Curso (TCC) apresentado ao curso de Sistemas de Informação do Centro Universitário Christus, como requisito parcial para obtenção do grau de bacharel em Sistemas de Informação.

Aprovada em:

BANCA EXAMINADORA

Prof. Dr. Gleidson Sobreira Leite (Orientador)
Centro Universitário Christus (Unichristus)

Prof. Me. Felipe Timbo Brito
Centro Universitário Christus (Unichristus)

Prof. Me. Mariana da Conceição Carneiro Araujo
Centro Universitário Christus (Unichristus)

AGRADECIMENTOS

Quero agradecer primeiramente a Deus e à minha família, que sempre estiveram comigo nos melhores e piores momentos. Também agradeço a todos que caminharam ao meu lado até o meu último semestre, tanto os amigos novos quanto as pessoas que se tornaram quase irmãos para mim. Quero expressar minha gratidão ao Sr. Gleidson Sobreira Leite, ao Sr. Felipe Timbó Brito e à Srta. Mariana Maria da Conceição Carneiro Araújo. Além disso, gostaria de agradecer aos professores dos semestres anteriores e aos que não estão mais na instituição. Todos foram importantes para essa jornada.

"Os gênios começam grandes obras, os trabalhores acabam-nas"

(Leonardo Da Vinci)

RESUMO

Com o crescimento da indústria de jogos eletrônicos e o respectivo aumento de oportunidades para atuação profissional no mercado, cada vez mais surgem oportunidades para atuar no desenvolvimento de jogos. No entanto, verificam-se a existência de desafios a serem enfrentados tanto no aspecto técnico do desenvolvimento quanto na questão da atratividade e usabilidade. Para a criação de um jogo, é necessário passar por várias etapas, desde a ideia até o produto final. Nesse contexto, com o intuito de contribuir para a área acadêmica e mercadológica do desenvolvimento de jogos digitais, e utilizando as tecnologias adotadas no mercado atual, neste trabalho será apresentada uma revisão da literatura sobre o desenvolvimento de jogos digitais, abordando também as etapas e tecnologias de desenvolvimento. Em seguida, será apresentado um jogo iterativo desenvolvido com as tecnologias selecionadas e descritas na revisão da literatura.

Palavras-chave: desenvolvimento. jogos digitais. unity.

ABSTRACT

With the growth of the electronic games industry and respective growth of opportunities for professional performance in the market, opportunities are emerging to work with game development. However, there are challenges to be faced both in the technical aspect of development and in the issue of attractiveness and usability, since, in order to create a game, it is necessary to go through several stages, from the idea to the product. Final. In this context and in order to contribute to the academic and marketing area of digital game development, and using technologies adopted in the current market, this work will present a literature review regarding the development of digital games, as well as development stages and technologies . Then an iterative game developed with selected technologies and described in the literature review will be presented.

Keywords: Development, Digital Games, Unity.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE SÍMBOLOS

API	Application Programming Interface
I.A	Artificial Intelligence
C#	Csharp
RPG	Role Playing Game

SUMÁRIO

1 INTRODUÇÃO

1.1 Contextualização e delimitação do tema

O desenvolvimento de jogos é uma área que vem ganhando cada vez mais destaque no mundo da tecnologia e do entretenimento. Com a crescente popularidade dos jogos digitais, muitos desenvolvedores têm se dedicado a criar novos jogos ou aprimorar os existentes. No entanto, desenvolver um jogo pode parecer uma tarefa complexa e desafiadora, principalmente para quem está começando na área. Segundo Sousa e Bittencourt (2020), a indústria de jogos digitais tem experimentado um crescimento exponencial nos últimos anos, com o mercado global movimentando bilhões de dólares anualmente. Conforme apontado por Bonani (2020), a receita da indústria também tem apresentado um crescimento constante (conforme mostrado na Figura 1), inclusive em comparação com as indústrias de cinema e música (conforme demonstrado na Figura 2).

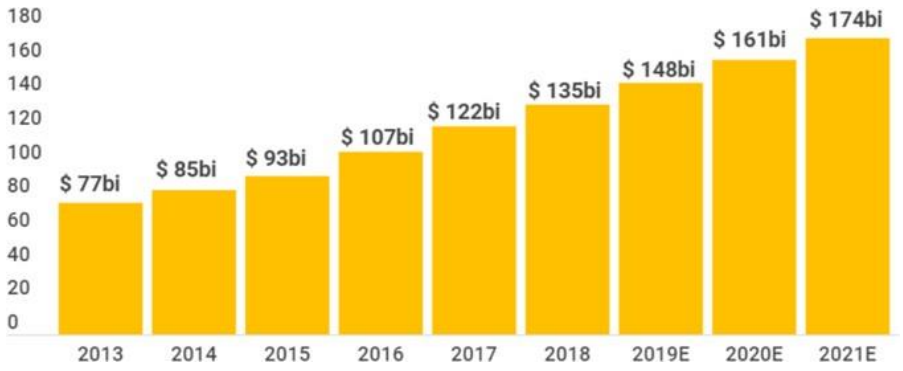

Figura 1: Tamanho da indústria em receita por ano em dólares [Bonani (2020)]

Figura 2: Receitas de videogames comparado com Cinema e Música [BONANI, 2020]

Assim, o mercado de jogos eletrônicos cresceu exponencialmente nos últimos anos e se tornou uma indústria global de bilhões de dólares. O mercado também experimentou um crescimento significativo no Brasil, com aumento do número de jogadores e lucratividade para os desenvolvedores de jogos. Diante dessa situação, é imprescindível que as empresas tenham uma metodologia de lançamento bem definida para garantir o sucesso do jogo e a satisfação dos jogadores. Nesse sentido, o estudo ESPM Brasil Gamer Profile fornece informações importantes sobre as preferências dos jogadores brasileiros por e-games, que podem ser usadas para desenvolver uma maneira eficaz de lançar jogos no mercado brasileiro (ESPM, 2021).

Dessa forma, é possível perceber que o desenvolvimento de jogos vem se tornando uma área de atuação cada vez mais importante e presente no dia a dia das pessoas, o que também impacta no crescimento de oportunidades para profissionais do ramo, como desenvolvedores, designers, músicos, artistas e demais profissionais.

1.2 Problematização e Objetivos

No contexto de desenvolvimento de jogos, segundo Campos (2017), um dos maiores desafios para os desenvolvedores de jogos é criar um jogo atraente e envolvente para o usuário, que prenda sua atenção e desperte seu interesse. Isso requer não apenas um bom conhecimento técnico, mas também uma compreensão das necessidades e exigências do público-alvo, bem como a capacidade de inovar e gerar soluções criativas.

Nesse cenário, verificam-se a existência de desafios a serem enfrentados tanto no aspecto técnico do desenvolvimento quanto na questão da atratividade e usabilidade, uma vez que, para a criação de um jogo, é preciso passar por várias etapas, desde a ideia até o produto final.

De forma a contribuir para o ramo de desenvolvimento de jogos digitais e utilizando tecnologias adotadas no mercado atual, neste trabalho será apresentado um jogo interativo, detalhando suas funcionalidades e tecnologias utilizadas. Dessa forma, também se pretende fornecer uma visão geral do processo de desenvolvimento, ajudando na formação de novos desenvolvedores de jogos, ao proporcionar insights sobre os desafios e oportunidades envolvidos na criação de jogos digitais.

Além disso, espera-se que este trabalho possa inspirar novas ideias e soluções para o desenvolvimento de jogos, contribuindo para o avanço e inovação na indústria de jogos digitais.

1.2.1 Objetivo geral

Diante do crescimento da indústria de jogos, assim como das oportunidades que surgem com esse cenário, este trabalho tem como objetivo o desenvolvimento de um jogo interativo chamado Old Bones.

1.2.2 Objetivos específicos

Os objetivos específicos deste trabalho são:

• Revisão da literatura a respeito das principais tecnologias utilizadas no desenvolvimento de jogos digitais

• Selecionar as tecnologias a serem adotadas para o desenvolvimento do jogo

• Idealização do jogo, suas funcionalidades e definir uma mecânica de jogo que seja intuitiva, fácil de aprender e envolvente para o jogador.

• Execução das etapas de desenvolvimento do jogo abordando os desafios e oportunidades do processo de desenvolvimento.

• Apresentação do jogo e suas funcionalidades.

1.3 Justificativa

Motivado pela oportunidade de estudo em uma área em constante crescimento, como a indústria de jogos, que também apresenta oportunidades profissionais em expansão, a escolha de desenvolver um jogo interativo foi feita para este trabalho, como forma de contribuição tanto acadêmica quanto mercadológica.

1.4 Estrutura do trabalho

Este trabalho está estruturado da seguinte maneira: no capítulo 1, consta a introdução com contextualização, problematização, objetivos e justificativa. No capítulo 2, é apresentada a revisão da literatura das tecnologias selecionadas para o desenvolvimento do jogo. No capítulo 3, a metodologia é demonstrada. Enquanto no capítulo 4, o jogo é descrito, e, por fim, no capítulo 5, expõe-se a conclusão do presente trabalho.

2 REVISÃO DA LITERATURA

Este capítulo apresenta a revisão da literatura com informações relacionadas ao desenvolvimento de jogos e seus desafios, etapas de desenvolvimento de jogos, assim como as tecnologias a serem utilizadas para o desenvolvimento do jogo proposto neste trabalho.

2.1 Desenvolvimento de jogos e seus desafios

Segundo Santos e Silva Junior (2016b), o desenvolvimento de jogos é caracterizado como todo o processo de concepção de um jogo, desde o levantamento de requisitos e necessidades até o processo de design e implementação, finalizando com a avaliação e implantação do jogo. Esse processo pode envolver profissionais de diversas áreas, como design, programação e marketing. Para garantir que o jogo seja atraente, funcional e bem-sucedido dentro do mercado.

Um dos grandes desafios ao criar um jogo é evitar que ele se torne trivial e monótono. Segundo Andrade (2006), o conflito é um dos pontos mais importantes em jogos, uma vez que os jogadores buscam sempre desafios adequados. No entanto, o balanceamento dos jogos se torna um problema a ser solucionado, já que é preciso criar mecanismos que desafiem o jogador de maneira adequada, evitando tarefas triviais que possam entediá-lo ou tarefas impossíveis que possam frustrá-lo.

Segundo Harrison, A. (2018) o planejamento é fundamental para garantir que o projeto seja concluído dentro do prazo e do orçamento estabelecidos. Schell, J. (2015) acresenta que o conceito do jogo é a ideia central que guia todo o processo de desenvolvimento (SCHELL, 2015, p. 23).

De acordo com Bates (2015), o desenvolvimento de jogos é uma atividade cola- borativa que requer pessoas de diferentes disciplinas, como: programação, design, arte e som, para criar um jogo divertido e envolvente. Desse modo tal tarefa requer uma equipe miltidisciplinar. Além disso, é fundamental manter um processo de desenvolvimento bem definido e seguir as melhores práticas para garantir que o jogo seja lançado no prazo e com a qualidade desejada.

Segundo Schell (2019), a criação de jogos é uma arte que combina criatividade, tecnologia e negócios para criar jogos únicos e envolventes para os jogadores. Para criar um jogo de sucesso, é importante seguir um processo de desenvolvimento bem definido que inclua desdea definição do escopo do projeto até as atividades de lançamento e pós-lançamento. É possível encontrar várias plataformas para desenvolvimento de jogos, como, por exemplo: Construct,

GameMaker Studio, RPG Maker, Unreal Engine, entre outras. Conforme discriminado na Tabela 1 abaixo.

Ferramenta	Descrição	Site
Unity	Com suporte para várias plataformas, incluindo PC, consoles e dispositivos móveis, o Unity é um dos mecanismos de jogos mais populares da atualidade (Harrison, 2019).	https://unity.com/
Unreal Engine	A ferramenta de criação em 3D em tempo real mais aberta e avançada do mundo (Unreal Engine, 2023)	https://www.unrealengine.com/
GameMaker Studio	Usando a poderosa e flexível plataforma GameMaker Studio, os desenvolvedores podem criar jogos 2D de forma rápida e eficiente (Chen & Chen, 2020) .	https://www.yoyogames.com/gamemaker
Construct	Permite que os desenvolvedores criem jogos 2D complexos sem a necessidade de conhecimento avançado de programação. Exemplos de plataformas com essas características incluem Construct, GameMaker Studio e Unity (Harrison, 2018, p. 5).	https://www.construct.net/en
RPG Maker	Oferece uma interface visual fácil de usar e recursos avançados de criação de histórias, permitindo que os desenvolvedores criem jogos de RPG complexos sem a necessidade de conhecimento avançado de programação (Schell, 2015, p. 456).	https://www.rpgmakerweb.com/

Tabela 1: Exemplos de ferramentas de desenvolvimento de jogos

Para este trabalho, a plataforma utilizada no desenvolvimento do jogo proposto é Unity e a linguagem de programação C#.

Segundo Chen et al. (2016), a plataforma Unity e a linguagem de programação C# são amplamente utilizadas no desenvolvimento de jogos digitais, no qual destaca que a plataforma Unity tem sua popularidade devido à sua facilidade de uso, flexibilidade e recursos.

Zhang et al. e Santos (2016) ressaltaram que Unity e C# são eficazes no desenvolvimento de jogos educacionais e permitem a criação de jogos interativos e envolventes que ensinam conceitos de saúde. Já Abdulhamid et al. (2017), destacaram que o Unity é bem sucedido no desenvolvimento de jogos de realidade virtual devido à sua facilidade de uso, recursos e suporte multiplataforma. Esses recursos permitem que os desenvolvedores criem jogos imersivos e interativos para fins de aprendizado, entretenimento ou educação. Por isso, Unity e C# ainda são ferramentas populares para criar jogos digitais.

2.2 Etapas de desenvolvimento de jogos

Para o desenvolvimento de jogos, algumas fases importantes merecem destaque, entre elas:

- A fase de pré-produção
- Definição do escopo do projeto
- Elaboração do design do jogo
- Programação
- Criação Artística
- Produção de Áudio
- Planejamento, execução e registro de testes
- Correção dos problemas identificados
- Definição da plataforma e data de lançamento
- Criação de uma Campanha de Marketing
- Lançamento do Jogo e Monitoramento pós-lançamento

A fase de pré-produção é crucial para o sucesso de um jogo, pois é nessa etapa que são realizadas pesquisas de mercado e público-alvo que ajudam a definir o escopo do projeto e a identificar as necessidades dos jogadores. Segundo Schell (2019), a pré-produção é a fase em que se define o que se quer fazer e como fazer. Já na fase de criação, a equipe de desenvolvimento trabalha na concepção de todo o jogo, desde a geração de ideias até a implementação dos recursos.

Segundo Goldstone, W. (2011), a pré-produção é uma etapa crucial no desenvolvimento de jogos, pois permite que os criadores testem e aprimorem suas ideias antes de iniciar a produção. O autor acrescenta que a pré-produção é uma fase crítica do processo de desenvolvimento de jogos, pois possibilita aos desenvolvedores testar e fortalecer as ideias antes de começar a produção.

Durante a pré-produção, os desenvolvedores podem fazer pesquisas de mercado, criar protótipos e testar os conceitos e a mecânica do jogo. Isso ajuda a garantir que o jogo tenha boa jogabilidade e seja envolvente para o público-alvo. Segundo Harrison (2018, p.12), apré-produção é uma etapa crucial para garantir que o jogo tenha boa jogabilidade e seja atraentepara o público-alvo.

Além disso, a pré-produção ajuda a definir o escopo do projeto e estabelecer um cronograma de desenvolvimento realista. Isso é essencial para garantir que o projeto seja

concluído dentro do cronograma e das restrições orçamentárias estabelecidas. Segundo Schell (2015, p.23), a pré-produção é uma etapa crucial para definir o escopo do projeto e estabelecer um cronograma realista de desenvolvimento.

Com relação à definição do escopo do projeto, segundo Schell (2019), o escopo é uma definição do que o jogo é e do que não é. O escopo do projeto deve ser claramente definido nesta fase, levando em consideração o gênero do jogo, a plataforma de desenvolvimento, o público-alvo e os recursos disponíveis.

Na elaboração do design do jogo, de acordo com Bates (2015), a arte do design de jogos reside na criação de jogos que sejam interessantes, desafiadores e divertidos. Nesta fase, é extremamente importante desenvolver o design do jogo, que inclui a mecânica de jogo, a história, os personagens, os níveis e os gráficos.

Com relação à programação, segundo Adams e Dormans (2017), o software é a espinha dorsal do desenvolvimento de jogos porque permite que o jogo funcione corretamente. Nesta fase, a equipe de desenvolvimento trabalha na programação do jogo usando as linguagens de programação e as ferramentas de desenvolvimento adequadas.

Na criação artística, de acordo com O'Connor (2014), a arte é um componente fundamental do desenvolvimento do jogo, pois dá vida ao mundo do jogo e cria a atmosfera certa. Nesta fase, a equipe de desenvolvimento trabalha na criação dos visuais, modelos 2D/3D, texturas e animações do jogo.

Quanto à produção de áudio, Collins (2017) afirma que o som é um componente importante no desenvolvimento de jogos, pois ajuda a criar a imersão e a atmosfera adequadas. Nesta fase, a equipe de desenvolvimento trabalha na criação do áudio do jogo, que inclui diálogos, efeitos sonoros e trilhas sonoras.

Para a etapa de testes, aderir às melhores práticas de teste é fundamental para garantir a qualidade do jogo, como usar ferramentas de automação de teste e executar testes em várias plataformas e ferramentas. De acordo com Collins (2017), usar ferramentas de automação de teste é a melhor maneira de desenvolver jogos, pois permite que a equipe execute testes com mais eficiência e detecte problemas mais rapidamente. Além disso, testes em diferentes plataformas e gadgets são necessários para garantir que o jogo funcione bem em todas as condições.

De acordo com Bates (2015), o planejamento de teste é uma parte importante do desenvolvimento do jogo, pois permite que a equipe determine quais aspectos do jogo devem ser testados e como devem ser testados. Nesta etapa, é necessario planejar os testes a serem

realizados, como também definir cenários de teste e critérios de aceitação.

Quanto a execução dos testes, segundo Adams e Dormans (2017), o teste é uma parte importante do desenvolvimento do jogo, pois permite que a equipe identifique e corrija quaisquer problemas ou defeitos no jogo. Os testes desta fase são executados de acordo com o planejamento de teste definido na fase anterior.

No registro dos resultados do teste, O'Connor (2014) argumenta que o rastreamento dos resultados dos testes é fundamental para o desenvolvimento de videogames, pois permite que a equipe acompanhe o andamento dos testes e identifique quaisquer problemas ou defeitos que precisem ser corrigidos. Nesta etapa, os resultados dos testes são registrados em um relatório de teste, que contém informações sobre os testes realizados, os resultados obtidos e as correções feitas.

Quanto a correção dos problemas identificados, de acordo com Bates (2015), resolver os problemas encontrados durante o teste é um componente chave do desenvolvimento do jogo, pois permite que a equipe melhore a qualidade do jogo e forneça aos jogadores uma melhor experiência de jogo. Nesta fase, a equipe de desenvolvimento trabalha para resolver os problemas identificados durante o teste.

A etapa de definição da plataforma de lançamento é onde é definida a plataforma de lançamento do jogo, que pode ser console, desktop ou celular. Segundo o estudo Game Brasil do Grupo Sioux, os smartphones são a plataforma mais popular entre os gamers brasileiros, seguidos pelos PCs e consoles (Grupo Sioux, 2020).

Na fase da definição da data de lançamento, é muito importante determinar a data de lançamento do jogo, levando em consideração fatores como competição, feriados e eventos do setor. De acordo com o estudo Game Brasil (Grupo Sioux, 2020), a maioria dos gamers brasileiros prefere lançar jogos no segundo semestre do ano.

Na fase da criação da campanha de marketing, é o momento de criar uma campanha para promover o jogo e chamar a atenção dos jogadores. De acordo com o estudo Game Brasil, a maioria dos gamers brasileiros descobre novos jogos por meio de publicidade em redes sociais e plataformas de streaming (Grupo Sioux, 2020).

Na etapa do lançamento do Jogo, o jogo é lançado no mercado de acordo com as estratégias definidas nas fases anteriores. De acordo com o estudo Game Brasil (Grupo Sioux, 2020), a maioria dos gamers brasileiros opta por jogos gratuitos.

Por fim, na etapa de monitoramento pós-lançamento, é muito importante monitorar

o desempenho pós-lançamento do jogo, levando em consideração fatores como avaliação dos jogadores, vendas e atividade nas redes sociais. De acordo com o estudo Game Brasil (Grupo Sioux, 2020), a maioria dos gamers brasileiros prefere jogos com atualizações regulares e conteúdo adicional.

2.3 Unity

Uma das plataformas mais conhecidas e amplamente utilizadas para o desenvolvimento de jogos em todo o mundo é a Unity. Desde o seu lançamento em 2005, a Unity melhorou significativamente, expandindo suas funcionalidades e adaptando-se às mudanças na indústria de jogos (Goldstone, 2009).

David Helgason, Joachim Ante e Nicholas Francis fundaram a Unity em 2004 com o objetivo de criar uma plataforma acessível e simples de usar para desenvolvimento de jogos. A "comunidade"Unity desempenhou um papel crucial na evolução da plataforma. Os desenvolvedores compartilharam conhecimento, trabalharam juntos em projetos e ajudaram a plataforma a crescer por meio de fóruns, conferências e eventos (HELGASON, 2013).

Segundo Harrison (2018), devido à sua versatilidade e facilidade de uso, o Unity é uma das plataformas de desenvolvimento de jogos mais utilizadas no mercado. Os desenvolvedores podem criar jogos com Unity para uma variedade de plataformas, incluindo PCs, consoles, dispositivos móveis e realidade virtual (VR). Além disso, o Unity fornece uma ampla variedade de ferramentas e recursos para ajudar os desenvolvedores a criar jogos de alta qualidade rapidamente.

Com relação a alguns dos recursos da ferramenta, podemos citar:

1. Mecanismo de renderização: o Unity possui um mecanismo de renderização avançado que oferece suporte a gráficos 2D e 3D, permitindo que os desenvolvedores de jogos criem jogos visualmente impressionantes. Para melhorar a qualidade visual dos jogos, a plataforma também suporta shaders personalizados e técnicas de pós-processamento (FERRARI et al., 2018).

2. Programação: C# é a linguagem de programação que Unity usa para seus scripts e lógica de jogos. C# é simples de aprender e oferece recursos poderosos, incluindo gerenciamento de objetos, gerenciamento de memória e suporte para bibliotecas externas (HOCKING, 2018).

3. Armazenamento de ativos: O Unity possui uma loja de ativos integrada chamada Asset Store, onde os desenvolvedores podem comprar e vender itens, incluindo modelos 3D,

texturas, sons e scripts. Isso permite que os desenvolvedores economizem tempo e esforço comprando ativos prontos para uso em seus projetos (GOLDSTONE, 2011).

4. Ferramentas de animação: O Unity oferece ferramentas de animação integradas que permitem aos desenvolvedores criar animações complexas para gráficos e objetos de jogo (HARRISON, 2018).

5. Física e colisão: O Unity possui um sistema de física integrado baseado no conhecido mecanismo de física NVIDIA PhysX. Isso permite que os desenvolvedores de jogos criem interações realistas entre objetos e formas, incluindo detecção de colisão, modelagem de texturas e dinâmica de fluidos (JACKSON, 2014).

6. Plataforma cruzada: O Unity pode criar jogos para uma variedade de sistemas operacionais, incluindo Windows, macOS, Linux, Android, iOS, consoles de jogos e dispositivos de realidade aumentada e virtual. Isso facilita a distribuição do seu jogo em diversos mercados e dispositivos (HOCKING, 2018).

7. Documentação e tutoriais: O Unity oferece uma variedade de documentações e tutoriais para ajudar os desenvolvedores a aprenderem e aprimorarem suas habilidades na plataforma. Além disso, uma comunidade ativa e prestativa de desenvolvedores Unity compartilha informações e recursos por meio de fóruns e grupos online (GOLDSTONE, 2011).

Segundo o manual Unity User handbook (2019.4 LTS), o Unity é um dos engines (motor de jogos) de desenvolvimento mais utilizados, ou seja, um software especial com rotinas de programação para criar, planejar e operar ambientes interativos como jogos digitais, experiên- cias digitais, filmes ou televisão animada. Unity é uma plataforma de desenvolvimento com todos os recursos para criar jogos 3D e 2D multiplataforma, bem como experiências e simulações interativas. Com a ferramenta, você tem acesso a todo um ecossistema de desenvolvimento de jogos e pode facilmente criar, construir e implantar jogos em várias plataformas diferentes.

Segundo Gregory (2009), um motor de jogos é um software que ajuda a criar jogos e outras experiências digitais. Existem muitos tipos diferentes de motores de jogos, desde aqueles simples que ajudam a programar jogos menos complexos, até aqueles mais complexos que contêm ferramentas sofisticadas para facilitar a criação de jogos AAA (jogos com grandes orçamentos). Em geral, os motores são usados para ajudar a desenvolver jogos ou adaptá-los para diferentes plataformas.

Conforme apontado por El-Nasr et al. (2014), os principais componentes de um motor de jogo incluem um mecanismo gráfico para renderizar gráficos 2D e 3D, um mecanismo

físico para simular interações entre objetos, um sistema de animação para controlar personagens e objetos animados, um mecanismo de som para processar efeitos sonoros e música, e um sistema de inteligência artificial que controla personagens não-jogadores. Assim, existem várias maneiras diferentes de criar um motor de jogo, cada uma com suas próprias vantagense desvantagens. As funcionalidades típicas de um motor de jogo incluem motor gráfico para renderização de gráficos 2D e 3D, motor de física para simulação de interações entre objetos, sistema de iluminação, animações, sons, programação de inteligência artificial, programação por meio de scripts, simulação de partículas e muito mais.

Além disso, o Unity é utilizado em diversos setores no Brasil, incluindo arquitetura, engenharia e design. Um exemplo é o uso do Unity para criar simulações educacionais na indústria de óleo e gás (SANTOS et al., 2018). De acordo com o relatório da Newzoo de 2020, o Unity é a plataforma mais utilizada no Brasil para o desenvolvimento de jogos, com uma participação de mercado de 47%. Isso evidencia a importância do Unity na indústria de jogos brasileira e o amplo suporte que recebe por parte dos desenvolvedores.

2.4 C# no Unity

Segundo Will Goldstone em "Unity 3.x Game Development Essentials", a linguagem C# é uma das principais linguagens de programação usadas no Unity e é uma linguagem de programação orientada a objetos.

Além disso, Goldstone enfatiza que os scripts C# são fáceis de escrever e podem ser usados para controlar praticamente qualquer aspecto do jogo, e de acordo com a Microsoft (2021), C# é uma linguagem forte e flexível que simplifica a criação de jogos complicados. Além disso, o Unity fornece uma ampla gama de ferramentas e recursos para a criação de jogos em C#, como um editor visual, um depurador e uma biblioteca de scripts.

O uso de C# com Unity traz vários benefícios para o desenvolvimento de jogos. Segundo Hocking (2018), um dos principais benefícios é a facilidade de uso da linguagem que permite que desenvolvedores novatos e experientes criem jogos complicados com facilidade. Além disso, a linguagem de programação C# é forte e flexível, possibilitando o desenvolvimento de jogos com alto desempenho e qualidade. Outro benefício de usar C# com Unity é a grande comunidade de desenvolvedores que usam a linguagem. De acordo com Harrison (2018), isso significa que existem muitos recursos, tutoriais e fóruns disponíveis para ajudar os desenvolvedores a lidar com problemas e aprender novas técnicas.

Para demonstrar a importância do C# no Unity, podemos citar vários jogos conhecidos criados usando essa combinação, como o jogo de plataforma Hollow Knight, o jogo de tiro em primeira pessoa Superhot e o jogo de aventura de mundo aberto Subnautica (Unity Technologies, 2021).

Algumas das principais bibliotecas e APIs do Unity para C# incluem:

Biblioteca física: Uma biblioteca física fornecida pela Unity ajuda os desenvolvedo- res a criarem jogos com interações realistas entre personagens e objetos. A biblioteca física no Unity é bastante poderosa, de acordo com Andrea Ferrari em "Unity 2018 Shaders and Effects Cookbook"(FERRARI et al., 2018).

Biblioteca de animação: o Unity possui ferramentas de animação integradas que permitem aos designers criar animações complexas para personagens e objetos do jogo. Will Goldstone enfatiza em "Unity 3.x Game Development Essentials"que a biblioteca de animação Unity é fácil de usar e permite que os desenvolvedores criem facilmente animações sofisticadas (GOLDSTONE, 2011).

API para desenvolvimento de interface de usuário: Unity tem uma API de desenvolvimento de interface de usuário que permite aos desenvolvedores criarem interfaces de usuário personalizadas para seus jogos e experiências interativas. A API da interface do usuário: para Unity é fácil de usar e permite que os desenvolvedores criem rapidamente interfaces de usuário personalizadas, de acordo com Simon Jackson em "Mastering Unity 2D Game Development"(JACKSON, 2014).

De acordo com Simon Jackson em "Mastering Unity 2D Game Development", o Unity fornece várias bibliotecas e API’s para ajudar os desenvolvedores a criarem jogos 2D envolventes em C#. Além disso, Jackson destaca a plataforma Unity poderosa, mas simples de usar, que permite a criação rápida de jogos e aplicativos interativos (JACKSON, 2014).

3 METODOLOGIA

Para atingir o objetivo deste trabalho, que consiste na criação de um jogo, foi realizada inicialmente uma revisão da literatura e de trabalhos relacionados ao desenvolvimento de jogos, tecnologias, metodologias, entre outros.

Na revisão bibliográfica, foram consultados livros, artigos científicos, publicações e sites oficiais, bem como outras fontes relevantes, a fim de obter um entendimento aprofundado sobre as origens, objetivos e princípios desses temas.

Embora tenha havido um estudo relacionado às possíveis tecnologias a serem abordadas no trabalho, a preferência pelo Unity ocorreu devido à grande influência que a disciplina de Jogos Digitais - que está na grade de disciplinas da Unichristus. – exerceu sobre a escolha. Nela, é lecionado Unity como base, além da já possuir familiaridade com a tecnologia. Tais pontos foram determinantes na tomada de decisão sobre qual tecnologia adotar.

Na pesquisa bibliográfica, segundo Marconi e Lakatos (2010), o pesquisador utilizou-se de obras já publicadas, como livros, artigos científicos, teses, dissertações e outros materiais de caráter científico ou sites oficiais, a fim de embasar teoricamente o estudo e obter informações relevantes sobre o tema em análise (MARCONI; LAKATOS, 2010).

Concluída a revisão da literatura e a seleção das tecnologias a serem adotadas, deu-se início às etapas de desenvolvimento do jogo. Para o desenvolvimento do jogo, as seguintes etapas foram executadas, seguindo as fases apontadas no item 2.2 da revisão da literatura:

Na fase de pré-produção: Nesse momento, definiu-se qual seria o estilo de jogo. A decisão tomada foi que seria um jogo de sobrevivência, com ondas de inimigos vindo na direção do jogador. A partir desse tema, foram buscados outros jogos com a mesma temática. Um dos jogos de referência foi o Vampire Survivor, no qual inimigos surgem em intervalos de tempo e o jogador precisa derrotá-los para sobreviver por 30 minutos. No jogo de referência, é possível aumentar de nível e obter novas armas. No entanto, o jogo não poderia ser exatamente igual, pois é necessário dar uma identidade própria para o jogo, caso se queira aumentar a probabilidade de sucesso. Foi, então, definida a ideia de um jogo ambientado no Velho Oeste, no qual mortos-vivos surgem do chão e o jogador precisa fugir e derrotá-los.

Com relação ao escopo do projeto: com a temática e o cenário de Velho Oeste definidos, foi necessário pensar nos detalhes do jogo. Como seria a mecânica para derrotar os inimigos? Decidiu-se que o jogador atiraria automaticamente balas com sua arma em intervalos regulares, sempre na direção em que estivesse olhando. Dessa forma, o jogador teria que estar

sempre de frente para o inimigo, mas também precisaria saber correr para evitar ser derrotado. A ideia dos inimigos era simples: eles andariam em linha reta em direção ao jogador, sem desviar de obstáculos, e caso atingissem o jogador, o mesmo perderia uma vida. Se todas as vidas fossem perdidas, o jogador perderia o jogo.

Na elaboração do design do jogo: como a temática era de Velho Oeste, utilizou-se o software Aseprite para criar as pixel art do projeto em geral. Foram utilizadas imagens de referência do deserto e estruturas antigas. A aparência do inimigo foi definida como uma caveira ambulante, e o chefe final seria uma caveira maior com uma coloração vermelha sobreposta. O personagem principal seria um xerife, de acordo com a temática, que carregaria seu revólver e dispararia periodicamente nos inimigos. Todos os efeitos sonoros, incluindo músicas, foram obtidos de um asset de sons da Unity. Os sons eram testados para verificar se estavam em harmonia com a temática do jogo.

A programação: Utilizou-se a linguagem C# para a programação e foram empregados sistemas de eventos para estabelecer as conexões entre os diversos eventos do jogo. Foram criadas classes chamadas "Handler"para serem os controladores principais de alto nível, como "CharacterHandler"ou "WeaponHandler".

Com ralação as etapas de testes: Os testes foram realizados de forma manual, nos quais a cada funcionalidade implementada, eram realizados testes manuais, buscando forçar erros afim de que se conseguisse deixar o jogo com o menor número possível de erros. Em seguida, o orientador e outros dois alunos do curso de Sistemas de informação também efetuaram testes e conseguiram identificar alguns problemas. Após reportarem para o autor deste trabalho suas indagações, o autor do trabalho efetuou os ajustes necessários.

Em virtude do trabalho ser voltado para fins acadêmicos, não foram executadas as etapas de lançamento, marketing e monitoramento pós-lançamento.

4 O JOGO OLD BONES

Neste capítulo serão apresentadas as funcionalidades e a mecânica do jogo desenvolvido.

4.1 Informações Gerais do Jogo

O jogo Old Bones é um jogo de ação e tiro, com um layout retrô e simplificado, focando em alta facilidade de uso para atrair os usuários a jogarem por períodos mais longos. O objetivo é desafiar o jogador a combater diversos personagens, como esqueletos, por exemplo, e a incentivar a progressão do jogador ao longo do tempo.

Durante o jogo, o usuário deve eliminar esqueletos que aparecem aleatoriamente, podendo obter itens em determinados momentos. Tal ítem prolonga a vida do personagem, proporcionando maiores chances, por parte do jogador, de sobreviver por mais tempo e obter pontuações mais altas.

A meta do jogo é alcançar a maior pontuação possível. Conforme o tempo passa, a dificuldade aumenta, com o aparecimento de chefões ou um número maior de esqueletos. Além disso, a arma do personagem principal é aprimorada, aumentando a quantidade ou velocidade de tiros.

A Figura 3 apresenta um exemplo da tela inicial do jogo, com um menu contendo informações sobre como jogar e iniciar o jogo..

Figura 3: Tela inicial do menu do jogo Old Bones

A Figura 4 apresenta a tela do jogo após seu início. Nessa tela é possível ver o coração na parte superior direita informando a quantidade de vidas do personagem, e no canto

superior esquerdo, retrata a quantidade de inimigos mortos e a pontuação do jogador. Mais centralizado na parte superior, consta a informação do tempo em que o jogador está naquele jogo, assim como os buffs que serão citados mais para frente do capitulo.

Figura 4: Tela inicial do jogo Old Bones

A Figura 5 apresenta um exemplo de tela do personagem principal atirando com a bala (em amarelo), e um exemplo de esqueleto inimigo onde, quando o personagem principal obtêm êxito ao eliminar o esqueleto, o mesmo fica momentaneamente morto e com uma imagem diferente, conforme apresentado na Figura 6 antes de desaparecer.

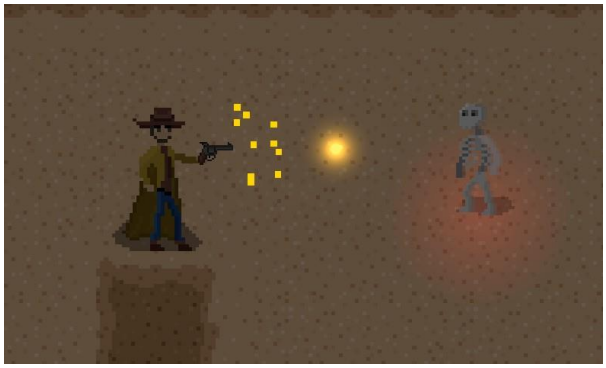

Figura 5: Exemplo de momento em que personagem atira em esqueleto inimigo

Figura 6: Exemplo ilustrativo de esqueleto inimigo após ser eliminado e antes de desaparecer

Para aumentar a possibilidade de jogar por mais tempo e obter maiores pontuações ao enfrentar os desafios do jogo, foi adicionado um "coração"ao mapa. Esse elemento, comumente utilizado em jogos para representar vida ou saúde, permite que o jogador seja atacado pelos esqueletos e ainda sobreviva. A quantidade de vidas é aumentada à medida em que o personagem principal coleta o "coração"e é reduzida à medida em que o esqueleto toca no jogador. Na Figura 7 apresenta, à esquerda, um exemplo de coração disponível para ser coletado pelo personagem principal. À direita, o ítem visível no canto superior direito da tela do jogador, exibe o contador de vidas do personagem.

Figura 7: Exemplo do item de vida para coleta pelo personagem e exemplo de contador de vidas

Para oferecer maiores desafios aos jogadores, também foram adicionados personagens mais fortes e difíceis de serem eliminados, chamados de "chefões". Esses chefões aparecem a cada minuto e apenas um deles pode estar vivo simultaneamente. A Figura 8 apresenta um exemplo de imagem de um chefão.

Figura 8: Exemplo de um chefão

Após o personagem principal obter êxito na eliminação do chefão, ele ganha benefícios aleatórios, como redução do tempo de delay entre os tiros (o jogador atira mais rápido) ou aumento do número de balas simultâneas. Os benefícios são concedidos aleatoriamente ao jogador a cada morte de chefão, mas as reduções normalmente são de 1 segundo por tiro, podendo chegar a um mínimo de 0.6 segundos por tiro. Já o número máximo de balas simultâneas é de 3 tiros.

Também foi implementado um sistema de pontuação, no qual cada chefão eliminado pelo jogador dobra a pontuação dos esqueletos. Se o jogador entrar em contato com algum

inimigo e ainda tiver "vidas"restantes, a pontuação é decrementada e um benefício é retirado aleatoriamente.

Caso o personagem principal não tenha mais vidas e entre em contato com algum inimigo, o jogo acaba e a tela de "Fim de Jogo", ilustrada na Figura 9, é apresentada ao usuário. Há também a exibição de informações como a pontuação obtida, a quantidade de inimigos eliminados e o tempo de jogo.

Figura 9: Exemplo de tela de “Fim de Jogo”

A seguir, serão apresentadas informações detalhadas sobre a implementação, incluindo descrições gerais e informações de código categorizadas por dados do jogador, geração de inimigos e status do jogador.

4.2 Informações do Jogador

O personagem principal periodicamente faz uso da sua arma para disparar balas. As munições que se acertam o inimigo, causam dano. O intervalo entre cada tiro é definido pela variável “timeToShoot”, com os upgrades é possível diminuir esse tempo (até um limite pré-definido). A Figura 10 apresenta parte do código dentro do método Update.

```
_actualTimeToShoot -= Time.deltaTime;
if (_actualTimeToShoot <= 0f)
{
    StartCoroutine(Shoot());
    _actualTimeToShoot = _timeToShoot + (_buff.TimeReductionQuantity * _buff.TimeReductionTimes);
}
```

Figura 10: Código do tempo de tiro

Conforme pode ser percebido, existe outra variável chamada "buff"que contém as informações dos upgrades que o jogador possui no momento (que serão detalhadas posteriormente). Quando o temporizador zera, o tiro é efetuado e a contagem recomeça. A Figura 11 apresenta

um exemplo de tela de configuração do prefab (objetos pré-fabricados que podem ser utilizados várias vezes dentro do jogo) da bala a ser disparada, onde são armazenadas basicamente a velocidade com que a bala viaja (igual à movimentação do inimigo, multiplicada pela variável "Speed") e o dano que ela causa, deixado por padrão no valor de 1.

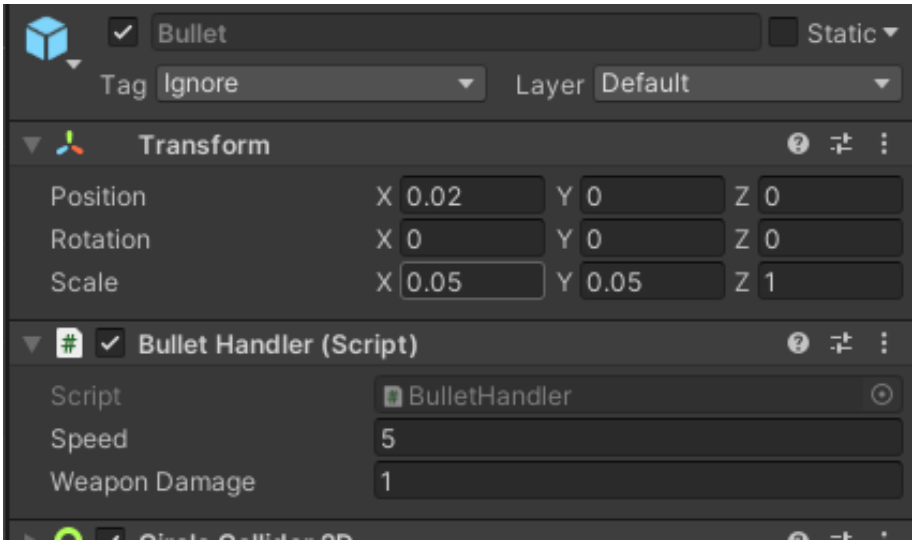

Figura 11: Configuração da bala a ser disparada pelo personagem

A movimentação do jogador é um processo bastante simples, onde os inputs do jogador são recebidos no método "Update()"para verificar se ele está se movendo em algum eixo (horizontal ou vertical). A Figura 12 ilustra parte do código que lida com a recepção das entradas do usuário.

```
};

_horizontal = Input.GetAxis("Horizontal");
_vertical = Input.GetAxis("Vertical");
```

Figura 12: Código de recebimento de entrada de usuário

4.3 Movimentação do Jogador

Conforme exemplificado na Figura 13, a movimentação do personagem é realizada ajustando a propriedade "RigidBody"(uma propriedade nativa da Unity usada para controlar movimentos físicos). No entanto, antes de fazer isso, é verificado se o personagem está vivo (por meio da variável "isAlive") e se o jogo não está pausado. Caso o personagem não esteja vivo ou o jogo esteja pausado, o método é encerrado para evitar que o jogador mova o personagem em situações indesejáveis, como quando o personagem está morto mas continua se movimentando.

```
Mensagem do Unity | 0 referências
private void FixedUpdate()
{
    if (!_isAlive || _menuHandler.IsGameActive() || SaveHandler.Instance.GamePaused)
    {
        _rigidBody.velocity = Vector3.zero;
        return;
    }

    if (_horizontal != 0f || _vertical != 0f)
    {
        _rigidBody.velocity = new Vector2(_horizontal * _moveSpeed, _vertical * _moveSpeed);
        if (Mathf.Abs(_horizontal) > 0.2f || Mathf.Abs(_vertical) > 0.2f) _oldPosition = new Vector3(_horizontal, _vertical, 0);
    }
    else
    {
        _rigidBody.velocity = Vector2.zero;
    }
}
```

Figura 13: Código de função FixedUpdate

Caso o personagem sofra dano do inimigo, as consequências são tratadas pelo método “DamagePlayer()”, conforme exemplificado na Figura 14:

```
1 referência
public void DamagePlayer(int damagePoints)
{
    if (!_canTakedamage) return;
    _lifePoints -= damagePoints;
    UpdateLifePointsText();
    if (_lifePoints <= 0)
    {
        KillPlayer();
    }
    else
    {
        _canTakedamage = false;
        _audioSource.PlayOneShot(_damageClip, 20);
        _buff.UnupgradeRandom();
        StartCoroutine(FadeInFadeOut());
    }
}
```

Figura 14: Código de função DamagePlayer

Conforme exemplificado na Figura 14, primeiramente verificamos se o personagem pode tomar dano. Caso seja negativo, o método é encerrado. Essa validação é realizada para evitar que o personagem morra imediatamente ao entrar em contato com um inimigo. Durante um período em que o personagem está piscando na tela após sofrer dano, ele se torna invulnerável e não receberá dano mesmo se entrar em contato com inimigos ou sofrer ataques. Em seguida, a vida do personagem é reduzida e a informação é atualizada na tela do jogador por meio do método "UpdateLifePointsText()". Uma validação adicional é realizada:

1. Se a vida do jogador for menor ou igual a zero, o método "KillPlayer()"é chamado para eliminar o personagem.

2. Caso contrário, o som de dano do personagem é reproduzido, e um dos upgra- des obtidos durante o jogo é removido aleatoriamente.

A Figura 15 exemplifica o método "KillPlayer()".

```
Mensagem do Unity | 0 referências
private void FixedUpdate()
{
    if (!_isAlive || _menuHandler.IsGameActive() || SaveHandler.Instance.GamePaused)
    {
        _rigidBody.velocity = Vector3.zero;
        return;
    }

    if (_horizontal != 0f || _vertical != 0f)
    {
        _rigidBody.velocity = new Vector2(_horizontal * _moveSpeed, _vertical * _moveSpeed);
        if (Mathf.Abs(_horizontal) > 0.2f || Mathf.Abs(_vertical) > 0.2f) _oldPosition = new Vector3(_horizontal, _vertical, 0);
    }
    else
    {
        _rigidBody.velocity = Vector2.zero;
    }
}
```

Figura 15: Código de função KillPlayer

Conforme apontado na Figura 15, a função remove as propriedades físicas do personagem, impossibilitando o corpo de se mover mesmo que o jogador continue apertando as teclas. É atribuído o valor false para a variável “isAlive” e é ativado o painel de fim de jogo.

4.4 Geração de inimigos

A geração de inimigos é controlado pelo script “SpawnEnemies.cs” (ver Figura 17). O jogo possui uma variável do tipo float que seta o intervalo em segundos dos inimigos que serão gerados no mapa (essa variável tem seu valor alterado, diminuindo à medida que o tempo que o jogador permanece vivo aumenta. Ou seja, quanto maior for o tempo em que o jogador permanece vivo e jogando, maior será a frequência de inimigos nascidos no mapa). Uma vez que a variável chega a zero, inimigos são gerados pelo mapa e então seu contador reseta (ver Figura 16).

```
        _actualTimeToSpawn -= Time.deltaTime;

        if (_actualTimeToSpawn <= 0)
        {
            SpawnEnemie();
            _actualTimeToSpawn = _timeToSpawn;
        }
```

Figura 16: Código do Tempo de geração de inimigos

Para gerar inimigos, é chamado a função “SpawnEnemie” (conforme identificado na Figura 16), esse por sua vez pode receber como parâmetro se ele deve ou não gerar “chefões”. Caso já tenha um chefe vivo no momento e seja solicitado que gere um novo chefe, nada acontece, pois segundo as regras do jogo, não se pode ter dois chefes em campo ao mesmo tempo. A Figura 17 exemplifica a função “SpawnEnemie”.

```
private void SpawnEnemie(bool spawnBoss = false)
{
    if (spawnBoss && _isBossAlive)
    {
        return;
    }
    for (int i = 0; i < (_quantityToSpawn + (spawnBoss ? 0 : Mathf.Floor(_minutesPassed / 2))); i++)
    {
        Vector3 spawnPosition = Utils.SpawnPositionRelativeToPlayer(5,5);
        Instantiate(spawnBoss ? _bosses[0] : _enemies[0], spawnPosition, Quaternion.identity);
    }
}
```

Figura 17: Código de função SpawnEnemie

O local onde os inimigos serão gerados é definido pela função “SpawnPositionRelativeToPlayer” (ver Figura 18), dentro do script “Utils.cs”. Esse, por sua vez, recebe duas variáveis do tipo inteiro (conforme exemplificado na Figura 17 que demonstra usando o valor 5 para as duas variáveis) para definir a distância nos eixos X e Y - aceitável para a geração do inimigo. Afunção pega essas distâncias e gera um ponto aleatório nos eixos X e Y e verifica se esse pontonão tem nada. Caso positivo, temos um local possível para gerar inimigo, caso contrário, ele entra em recursividade e se chama novamente, tentando gerar outro ponto aleatório, até que eleconsiga achar um ponto disponível.

```
3 referências
public static Vector3 SpawnPositionRelativeToPlayer(int distanceX, int distanceY)
{
    Vector3 spawnPosition = new Vector3(CharacterHandler.Instance.gameObject.transform.position.x + RandomNumber(distanceX),
        CharacterHandler.Instance.gameObject.transform.position.y + RandomNumber(distanceY), 0);
    var position = Physics2D.OverlapPoint(spawnPosition);
    if (position == null)
    {
        return spawnPosition;
    }
    else
    {
        return SpawnPositionRelativeToPlayer(distanceX, distanceY);
    }
}
```

Figura 18: Código geração inimigo.

Dessa forma, temos o controle da geração de inimigos. A variável que determina o intervalo de tempo para a geração dos inimigos ("timeToSpawn") é declarada como privada, mas possui a marcação "[SerializeField]". Isso permite que seja possível alterar diretamente no editor da Unity o tempo em que novos inimigos serão gerados.

4.5 Código dos Esqueletos

O código dos esqueletos funciona de maneira padronizada, onde os inimigos sempre perseguem o jogador, percorrendo o menor caminho até ele (ou seja, sempre andam em linha reta até o jogador). Não desviam de obstáculos, não desviam de tiros, sempre com o intuito de encostar no jogador. Caso o esqueleto encoste no jogador, ele sofre dano. Caso o jogador não tenha vidas suficientes, o jogador perde a partida. Função exemplificada na Figura 19.

```
Mensagem do Unity | 0 referências
void Update()
{
    if (_isSpawning)
    {
        return;
    }
    if (SaveHandler.Instance.GamePaused)
    {
        return;
    }
    if (!CharacterHandler.Instance.IsAlive())
    {
        _isMoving = false;
        if(_rb != null)
        {
            _rb.velocity = Vector3.zero;
        }
        return;
    }

    _direction = _player.position - transform.position;
    _direction.Normalize();

    if (_player != null) _isMoving = true;
    else _isMoving = false;

    _scorePoints = _isBoss ? EnemyValuePoints.SkeletonBossPoints : EnemyValuePoints.SkeletonPoints;
}
```

Figura 19: Código de função de direcionamento de inimigos

Os inimigos recebem na hora de serem gerados o “transform.position” do personagem (propriedade que informa a posição do jogador nos eixos X, Y e Z, onde o Z é descartado porque estamos tratando de um jogo 2D, ficando somente com os eixos X e Y). Dessa forma, é calculado a direção no qual eles devem andar até o jogador no método “Update” exemplificado na Figura 19 (chamado a cada frame do jogo) e no método “FixedUpdate” (também chamado a cada frame, porém com tratativas melhores quando é usado para a física no jogo) movemos o inimigo até a direção selecionada. A Figura 20 apresenta o método “FixedUpdate”.

```
Mensagem do Unity | 0 referências
private void FixedUpdate()
{
    if (_isSpawning)
    {
        return;
    }
    if (_player != null && _isAlive && CharacterHandler.Instance.IsAlive())
    {
        _rb.MovePosition((Vector2)transform.position + ((Vector2)_direction * _speed * Time.deltaTime));
    }
}
```

Figura 20: Código de função FixedUpdate

Vale observar que no código, existe uma variável chamada "isSpawning". Caso essa variável seja verdadeira (true), o método é encerrado sem executar o restante do escopo. Isso ocorre porque quando um inimigo é gerado, há um contador para controlar sua inatividadeinicial (usado, por exemplo, para reproduzir a animação de geração do inimigo e evitar que o personagem morra instantaneamente por causa de um inimigo gerado em sua frente). Após a contagem regressiva, o inimigo está livre para agir de acordo o programado. A mesma lógica se aplica aos chefes, porém eles possuem uma quantidade maior de vida.

4.6 Status dos Inimigos

Os inimigos iniciam com uma quantidade “X” de vida. Conforme recebem tiros do jogador, eles vão perdendo pontos de vida e quando a quantidade chega a zero, é executado o som da animação de morte e o jogador recebe sua pontuação. Caso o inimigo seja um chefe, a pontuação é maior e as futuras pontuações são aumentadas, além de receber um upgrade aleatório (seja para causar mais dano por tiro, ou atirar mais vezes). A Figura 21 apresenta a tela de configuração do esqueleto inimigo.

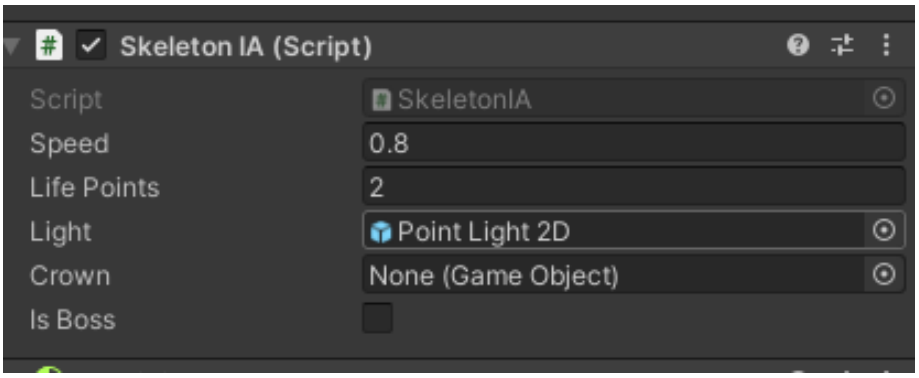

Figura 21: Código de configuração do esqueleto inimigo

Os inimigos são uma prefab que são instanciadas uma vez quando o contador é zerado. Fazendo uso do editor, é possível alterar a quantidade de vida do inimigo e sua velocidade de movimento (também existe a prefab do chefe, onde a vida é maior). A Figura 22 exemplifica o código utilizada para casua do dano (“Damage”).

```
2 referências
public void Damage(int damagePoints)
{

    if (!_isAlive) return;

    _lifePoints -= damagePoints;
    if(_lifePoints <= 0)
    {
        _isAlive = false;
        Die();
        OnDie?.Invoke(this, EventArgs.Empty);
    }
    else
    {
        OnDamage?.Invoke(this, EventArgs.Empty);
        if (!_isSlowDown) StartCoroutine(SlowOnShoot());
    }
}
```

Figura 22: Código de função Damage

Vale acrescentar, também, que sempre que o inimigo sofre dano e não morre, é chamada a rotina de causar lentidão (ver Figura 23). É uma forma de feedback ao jogador, mostrando ao usuário que o inimigo sofreu dano, mas não morreu. Além de também chamar o evento “OnDamage”, exemplificado na Figura 24.

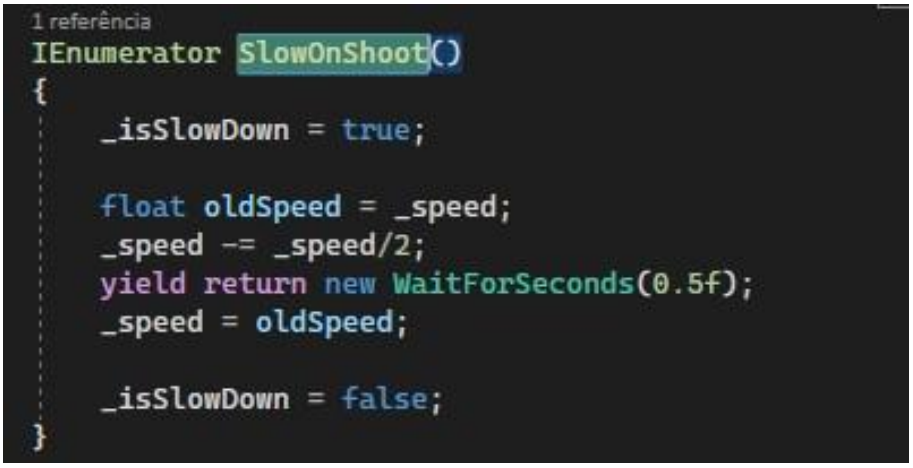

Figura 23: Código de lentidão após tomar dano

Figura 24: Código para fazer inimigo “piscar” na tela após tomar dano.

No evento "OnDamageEvent", ocorre outro feedback de dano caso o inimigo não morra. Nesse evento, é executado o som de dano ("damageClip") e a rotina "FadeInFadeOut()"é chamada (exemplificada na Figura 25), fazendo o inimigo piscar.

```
IEnumerator FadeInFadeOut()
{
    _isDamageCoroutineRunning = true;
    float elapsedTime = 0f;
    int counter = 0;
    bool fadingOut = false;

    while (elapsedTime < 0.1f && counter < 6)
    {
        if (!fadingOut)
        {
            elapsedTime += Time.deltaTime;
            _spriteRenderer.color = Color.Lerp(_initialColor, _fadeOut, elapsedTime / 0.1f);
            if (_spriteRenderer.color == _fadeOut)
            {
                fadingOut = !fadingOut;
                counter++;
                elapsedTime = 0f;
            }
        }
        else
        {
            elapsedTime += Time.deltaTime;
            _spriteRenderer.color = Color.Lerp(_fadeOut, _initialColor, elapsedTime / 0.1f);
            if (_spriteRenderer.color == _initialColor)
            {
                fadingOut = !fadingOut;
                counter++;
                elapsedTime = 0f;
            }
        }
        yield return null;
    }
    _isDamageCoroutineRunning = false;
}
```

Figura 25: Código de função FadeInFadeOut

Em caso de morte do inimigo, a função “IncreaseScore()” (exemplificada na Figura 26) é chamada, que é responsável basicamente por aumentar a pontuação do jogador:

```
1 referência
public void IncreaseScore(int amount)
{
    score += amount;
    deadEnemies += 1;
    UpdateScore();
}
```

Figura 26: Código de função IncreaseScore

Acrescenta-se que a função “IncreaseScore()” chama outra função chamada “UpdateScore()” (exemplificada na Figura 27). Essa é a responsável por atualizar a pontuação do jogador – seja para mais ou para menos.

```
4 referências
public void UpdateScore()
{
    _score.text = "Score: " + score;
    _enemiesKilledText.text = "Inimigos mortos: " + deadEnemies;
}
```

Figura 27: Código de função UpdateScore

5 CONCLUSÃO

Com o intuito de contribuir com a área acadêmica e mercadológica de desenvolvimento de jogos digitais, este trabalho apresentou uma revisão da literatura sobre conceitos relacionados ao desenvolvimento de jogos digitais, assim como de tecnologias e etapas de desenvolvimento.

Nesse contexto, foi apresentado um jogo interativo chamado Old Bones, desenvolvido com a plataforma Unity e a linguagem de programação C#. É possivel de ser baixado pelo link:<https://drive.google.com/file/d/1iWJg0mUwI9_4WO8ThVVvR5e_83zIVW-d/view?usp=sharing>

Apesar das dificuldades no transcurso desse trabalho, tendo em vista que o autor foi o único responsável pela execução das etapas de desenvolvimento do jogo, desde a pré-produção até a programação e implantação, registro aqui que a experiência e o aprendizado no decorrer do desenvolvimento do referido jogo foram de grande valia.

Como limitações do trabalho, pode-se destacar o fato de que a equipe de desenvolvimento foi de somente um único participante, sendo este o responsável por todas as etapas de desenvolvimento do jogo. O que acabou por impossibilitar a consecução das etapas com maior seletividade e rigor técnico. Outro ponto limitador na execução do trabalho aqui demonstrado, foi o fator tempo. Por se tratar de um cenário acadêmico, não visando o lançamento do jogo no mercado, foi preciso investir muito tempo em questões de formatação de trabalhos acadêmicos.

Outra limitação foi em relação à etapa de testes que envolveram apenas quatro pessoas: o autor, o orientador e outros dois alunos do curso de Sistemas de Informação da Unichristus.

Como trabalhos futuros, recomenda-se a implementação de novas funcionalidades no sistema, como diferentes tipos de inimigos com estratégias distintas, inimigos com ataques à distância, diferentes níveis de vida ou saúde, novos chefes, novos mapas e até mesmo um sistema de níveis.

Também é importante considerar a oportunidade de lançar o jogo para uma maior quantidade de usuários finais, visando o aprimoramento do jogo e a obtenção de contribuições mais abrangentes, além de ser possível obter feedbacks que são extremamente valorosos para a melhoria do jogo como um todo.

REFERÊNCIAS

ABDELHAMID, A. M.; ALIM, A. M.; KARRAY, K. E. A comparative study of game engines for virtual reality. In: IEEE. **2017 IEEE International Conference on Computational Science and Engineering (CSE) and IEEE International Conference on Embedded and Ubiquitous Computing (EUC)**, 2017. p. 159–164.

ADAMS, E.; DORMANS, J. **Game Mechanics: Advanced Game Design**. CRC Press,2017.

ANDRADE, G. D. **Balanceamento dinâmico de jogos: Uma Abordagem Baseada em Aprendizagem por Reforço**. Dissertação (Mestrado) — Centro de Informática, Universidade Federal de Pernambuco, Recife, 2006. [s.n.].

BATES, B. **Game Design: Principles, Practice, and Techniques**. CRC Press, 2015.

BONANI, B. **A Indústria de Games: Tudo o que você precisa saber para jogar certo – Resumo do Mercado**. 2020. Investing.com. Disponível em: <https://br.investing.com/analysis/a-industria-de-games-tudo-o-que-voce-precisa-saber-para-jogar-certo-200435456>.

CAMPOS, R. D. **Desenvolvimento de Jogos - Teoria e Prática**. Novatec Editora Ltda,2014.

CHEN, J.; CHEN, Y. A comparative study of game engines: Unity and unreal engine. In: IEEE. **2020 IEEE 3rd International Conference on Information Systems and Computer Aided Education (ICISCAE)**, 2020.

CHEN, J.; ZHU, Y.; MA, H.; XIE, C. Development of serious games using unity 3d: A review. In: IEEE. **2016 IEEE 18th International Conference on High Performance Computing and Communications; IEEE 14th International Conference on Smart City; IEEE 2nd International Conference on Data Science and Systems (HPCC/SmartCity/DSS)**, 2016. p. 1526–1531.

COLLINS, K. **Game Sound: An Introduction to the History, Theory, and Practice of Video Game Music and Sound Design**. MIT Press, 2017.

EL-NASR, M. S.; DRACHEN, A.; CANOSSA, A. **Game Analytics: Maximizing the Value of Player Data**. Springer, 2014.

ENGINE, U. **Unreal Engine**. <https://www.unrealengine.com/>. Acesso em: 07 maio 2023.

ESPM. **Perfil dos Jogadores de Games no Brasil**, 2021.

FERRARI, A.; CONTI, D.; SANTORO, C. **Unity 2018 Shaders and Effects Cookbook: Transform your game into a visually stunning masterpiece with over 70 recipes, 3rd Edition**. [S.l.]: Packt Publishing, 2018.

GOLDSTONE, W. **Unity Game Development Essentials**. Packt Publishing, 2009.

GOLDSTONE, W. **Unity 3.x Game Development Essentials**. Packt Publishing Ltd,2011.

GREGORY, J. **Game Engine Architecture**. A K Peters, 2009.

HARRISON, A. **Unity 2018 By Example: Learn about game and virtual reality development by creating five engaging projects, 2nd Edition**. Packt Publishing, 2018.

HARRISON, H. **Unity 2019 Game Development: Build 2D & 3D Games**. Packt Publishing Ltd, 2019.

Newzoo. **Brazilian Games Market**. 2020. <https://newzoo.com/insights/articles/brazilian-games-market/>.

O'CONNOR, S. **Level Up! The Guide to Great Video Game Design**. Wiley, 2014.

SANTOS, J. M.; SANTOS, J. C. D.; SOUSA, R. M. A comparative analysis of game engines for the development of serious games. In: IEEE. **2019 IEEE 6th Portuguese Meeting on Bioengineering (ENBENG)**, 2019.

SANTOS, R. A. *et al*. Desenvolvimento de simulações de treinamento para a indústria de petróleo e gás utilizando a unity. In: **Anais do XXVI Congresso Brasileiro de Engenharia Mecânica (COBEM)**. [S.l.: s.n.], 2018.

SOUZA, R. D. C.; BITTENCOURT, I. M. O processo de desenvolvimento de jogos digitais: uma revisão sistemática. **Research, Society and Development**, v. 9, n. 11, 2020. Disponível em: <https://rsdjournal.org/index.php/rsd/article/view/10327>.

Unity Technologies. **Unity User Manual (2019.4 LTS)**. 2019. <https://docs.unity3d.com/Manual/index.html>. Acesso em: 29 abril 2023.

Unity Technologies. **Unity**. 2021. <https://unity.com/>.

Unity Technologies. **Unity scripting API**. 2021. <https://docs.unity3d.com/ScriptReference/>.

ZHANG, Y.; WANG, H.; HU, X.; LIU, H. Design and development of an educational game for diabetes self-management. In: IEEE. **2016 IEEE Global Engineering Education Conference (EDUCON)**, 2016. p. 1453–1458.