Universidade Federal de Mato Grosso do Sul
Faculdade de Ciência da Computação

Alfredo Cáceres Bernal Júnior

Ferramenta de desenvolvimento de jogos “Unity”

Ponta Porã
2023

Alfredo Cáceres Bernal Júnior

Ferramenta de desenvolvimento de jogos “Unity”

Trabalho de Conclusão de Curso apresentado à Faculdade de Ciência da Computação da Universidade Federal de Mato Grosso do Sul para obtenção do título de Bacharel em Ciência da Computação.

Orientador: Professor Leonardo Souza Silva

Ponta Porã

2023

Ficha catalográfica
Universidade Federal de Mato Grosso do Sul

Alfredo Cáceres Bernal Júnior, 1999-

Ferramenta de desenvolvimento de jogos “Unity” / Alfredo Cáceres Bernal Júnior. - Ponta Porã, MS :[s.n.], 2023

Orientador: Leonardo Souza Silva

Trabalho de Conclusão de Curso (graduação) - Universidade Federal de Mato Grosso do Sul, Faculdade de Ciência da Computação.

1.Jogos eletrônicos. 2. Ferramenta de desenvolvimento. 3. Ciência da Computação. I. Silva, Leonardo Souza. II. Universidade Federal de Mato Grosso do Sul. III. Título.

Informações adicionais complementares

Título em outro idioma: Game development tool “Unity”
Palavras-chave em inglês:
Electronic Games
Development tool
Computer Science
Titulação: Bacharel
Banca Examinadora:
Leonardo Souza Silva

Data de entrega do trabalho em definitivo: 07/12/2023

Comissão Julgadora

Professor Leonardo Souza Silva

Orientador

Titular da Banca

Ponta Porã

2023

Dedico este trabalho aos meus pais e minha prima Ellen por não me fazer desistir.

Ponta Porã

2023

Agradecimentos

Agradeço aos meus pais Alfredo e Cristina por sempre me apoiarem e desejarem meu bem e por não me deixarem faltar nada.

Agradeço ao meu amigo Assem por sempre me ajudar a ficar com a mente mais leve, por sempre me ouvir e por me ajudar a construir um corpo mais saudável, melhorando minha atitude e auto-estima.

Agradeço a minha prima Ellen por me fazer voltar a Faculdade e terminar meus estudos e por me ajudar muito com a conclusão deste trabalho.

Agradeço a minha tia Kathia por me ajudar na formatação deste documento e a melhorar minha apresentação, além de todo apoio recebido.

Agradeço ao meu orientador Professor Leonardo por me dar clareza sobre que assunto devo defender e a iluminar meu caminho quando voltei a faculdade, e por ter a paciência de sempre tirar minhas dúvidas sobre o TCC.

Agradeço a todos meus professores e colegas por terem feito parte da minha formação acadêmica, por terem me ajudado nesta jornada e por serem gentis comigo.

Resumo

As *Game Engines* são ferramentas que auxiliam no processo de desenvolvimento de jogos, e podem servir para propósitos além desta indústria, porém muitos desenvolvedores que possuem o sonho de criar jogos autorais não possuem conhecimento dessas ferramentas e seus potenciais, ou usá-las, desistindo da carreira de desenvolvedor de jogos.

Este trabalho possui caráter híbrido, pois contém uma dissertação sobre o desenvolvimento de jogos e apresenta um estudo de caso sobre um jogo desenvolvido.

A ferramenta utilizada para desenvolver o jogo apresentado neste trabalho é a Unity, que apresenta aos desenvolvedores ferramentas poderosas e acessíveis para criar, operar e monetizar jogos em tempo real.

Palavras-chave: Desenvolvimento de jogos; Ferramenta Unity; Ciência da Computação.

Abstract

Game Engines are tools that simplify a lot of the process in game development and may serve purposes other than the gaming industry, but many with the dream of making their games don’t know their potential or how to use them, and that causes them to abandon those dreams.

This document has a hybrid character, as it has a dissertation on gaming development and presents a case study detailing a game developed to make the understanding more accessible and to add more knowledge on the subject.

The tool used to develop the presented game was the Unity tool, which gives developers powerful and accessible tools to create, operate and monetize games in real-time.

Keywords: Game development; Unity tool; Computer Science.

Lista de Siglas, Abreviações e Termos Gerais

Unity: Ferramenta utilizada para criar videogames e outros projetos 3D usando um editor.

COVID-19: É uma infecção respiratória causada pelo coronavírus e que causou uma pandemia e consequentemente um período de quarentena, que afetou o mundo todo e mudou muitas indústrias.

Mobile: Tecnologia que permite a pessoa poder se movimentar sem abdicar dela.

Jogador casual e *hardcore*: Um jogador é considerado casual quando alguém que gosta de jogar sem investir muito tempo, jogando espontaneamente e com pouca frequência.
Um jogador é considerado *hardcore* quando uma pessoa investe muito tempo jogando e possui conhecimento elevado sobre diversos jogos.

Cross-platform: Termo usado para descrever a possibilidade de adquirir e jogar um jogo em plataformas diferentes, por exemplo poder jogar o mesmo jogo em um PS4 e no PC.

Jogos de serviço: Termo para descrever um jogo que pode ser “alugado” usando uma assinatura de serviço, o exemplo mais popular é o *GamePass* do Xbox, no qual basta comprar uma assinatura para ter acesso a diversos jogos sem ter que comprá-los individualmente.

Arcade: Conhecidos no Brasil como Fliperamas.

Open-source: Significa código aberto, ou seja, quando uma aplicação tem seu código disponível para qualquer pessoa que pretende modificar e redistribuir o software.

RT3D: *Real Time* 3D, tecnologia que possibilita a manipulação de objetos 3D com atualização em tempo real.

VR e AR: Realidade Virtual e Realidade Aumentada.

2D e 3D: Duas dimensões e três dimensões.

VFX: Efeitos Visuais.

UI: Interface de Usuário.

MicroGames: Jogos que duram menos de 5 segundos. A Unity os utiliza para lhe ensinar como usar o editor.

Lista de Figuras

Sumário

1. Introdução

1.1 Contexto geral

Nos últimos anos o mercado de games teve um crescimento e está ocupando um grande espaço no mercado, e atualmente este setor lucra mais que as indústrias de música e cinema juntas (Wakka, 2021, p.1). Esses resultados são devido à uma grande quantidade de pessoas interessadas no mercado de games, a quantidade e a variedade de jogos lançados, e a quantia de dinheiro gasto em novos títulos e serviços de jogos existentes. Essas mudanças no mercado de games abre questionamentos como por exemplo, como os jogos estão sendo manuseados pelos usuários? qual a finalidade que esses jogos estão sendo desenvolvidos? e como esses jogos estão sendo monetizados?

1.2 Contexto específico

Um dos fatores que contribuiu com o crescimento da indústria dos videogames foi o isolamento que ocorreu devido a pandemia. Com isso, este mercado tende a dobrar de tamanho em seus ganhos até 2027, de acordo com uma nova reportagem da criadora da ferramentas de jogos Unity.
Uma figura que se destaca mais é a estimativa da Unity que a indústria dos jogos poderá ser de $300 bilhões de dólares até 2027 (Statt, 2022, p.1), graças a pandemia que induziu uma aceleração no número de pessoas que jogam, o números de jogos que são feitos e a quantia de dinheiro gasta em novos títulos e serviços de jogos existentes.
A reportagem, que tira dados de aproximadamente 230.000 desenvolvedores que usam a ferramenta Unity e outras, sublinha o quão drásticos foram os efeitos da COVID-19 na indústria de jogos, que viu uma grande 2020 e crescimento estável todo o ano de 2021 já que as pessoas ficaram dentro de casa e gastaram mais em entretenimentos de casa.

Muito do crescimento está ocorrendo no *mobile*, a reportagem diz. O número de jogos sendo desenvolvidos através de categorias desde casuais até os mais

hardcores está aumentando ao lado do número total de desenvolvedores, com um grande fluxo de novos jogos super casuais na categoria *mobile*.
"Publicadores na plataforma Unity construíram 93% mais jogos em 2021 do que em 2020 - isso é quase o dobro de jogos que o ano passado," disse a reportagem. "Há mais, tem mais criadores usando a plataforma Unity para construir jogos do que antes. Em 2021, o número de criadores Unity subiu 31% em comparação com 2020."(Statt, 2022)

A reportagem da Unity também destaca o crescimento e importância de *cross-platform* e jogos de serviços, duas tendências que a empresa diz que serão pilares enormes de como jogos serão criados e que títulos mantém popularidade e sucesso financeiro no futuro.
Graças a esse grande crescimento, muitas pessoas vieram a ter interesse no desenvolvimento de jogos e ir atrás de conhecer as qualificações necessárias para se criar um jogo e informações sobre o mercado de jogos.

1.3 Efeito adverso

Muitos destes interessados não possuem ideia de como começar ou qual a melhor ferramenta a utilizar. Neste contexto, cresce o interesse pelas "*Game Engines*" (Mecanismo de jogos), que são ferramentas utilizadas para facilitar o desenvolvimento de jogos e apoiar o desenvolvedor de diversas outras maneiras.

1.4 Causas

Uma das causas que levam a dificuldade dessa jornada de aprendizagem está no fato de que muitos dos requisitos não estão disponíveis como disciplinas em cursos, ou seja, não há disciplinas disponíveis que ensinam sobre desenvolvimento de jogos.
Outro grande obstáculo que muitos encontram é sobre os materiais de aprendizado estarem na língua inglesa, o que dificulta a jornada de muitos.

Outro fator que pode entrar em consideração é o equipamento necessário para desenvolver jogos. Para criar um jogo de qualidade gráfica alta é preciso um equipamento de custo elevado, porém, jogos com gráficos e visuais mais simples podem não precisar de um investimento tão alto e muitos podem não ter conhecimento sobre essa informação.

1.5 Objetivo

O objetivo deste trabalho é ajudar a todos que buscam conhecimento sobre desenvolvimento de jogos, mas estão perdidos sobre como começar e podem vir a pensar que é preciso muito conhecimento prévio e investimento.
Será apresentado um estudo sobre a ferramenta de jogos denominados de “Unity”, explicando seus pontos principais e metodologia. E um protótipo de jogo criado com base no que foi ensinado nas aulas da Unity.

2. Fundamentação Teórica

2.1 Sobre Vídeo Games

Entende-se por videogame qualquer jogo que possui um elemento interativo e mostra resultados da ação de um jogador em um display. Se usarmos esta lógica, pode-se dizer que os primeiros videogames surgiram no início da década de 50 e estavam altamente ligados com projetos de pesquisa universitária e de grandes corporações.

2.1.1 Origem dos vídeo games

Um dos primeiros jogos eletrônicos para entretenimento que pode-se dizer ter conhecimento surgiu em 1958, criado pelo físico Willy Higinbotham, que desenvolveu um "jogo de tênis" que se realizava em um osciloscópio e sendo processado por um computador analógico. O jogo foi batizado de "***Tennis Programming***", mas ficou mais conhecido por ***Tennis for Two***.
Antes disso os jogos eram criados para demonstrar uma tecnologia ou auxiliar em pesquisas, alguns do primeiros incluem ***Nimrod*** em 1951 (para jogar um jogo matemático chamado "Nim") por John Benett, ***OXO*** em 1952 (para simular o jogo da velha) por Alexander Douglas, e ***Hutspiel*** em 1955 (simulação militar).

Em 1966, o engenheiro Ralph Baer, acabou criando um jogo ao tentar criar um aparelho televisor revolucionário. Mais tarde essa invenção viria a se chamar de console.
Porém o projeto foi engavetado, pois não era o resultado que lhe foi solicitado pela ***Sanders Associates*** (Somente em 1971 foi retomado, quando a empresa Magnavox interessou-se pelo projeto, pelo jogo eletrônico).
E em 1967, na primeira atualização do projeto de Baer, é lançado uma espécie de jogo de *ping pong* para ser usado simultaneamente por até dois usuários.
Com isso, em 1968, Baer apresentou o requerimento da patente de seu protótipo de jogo eletrônico, rebatizado com o nome ***Brown Box***. A partir desse projeto, a Magnavox desenvolveu o ***Odyssey 100***, que se tornou o primeiro console de jogos eletrônicos dando início a era comercial dos jogos eletrônicos.

2.1.2 Como ficaram populares

Com o primeiro console, o ***Magnavox Odyssey***, foi marcado o início da primeira geração de consoles de videogame. Este console em si não ganhou muita popularidade, porém, influenciou diretamente a indústria de jogos arcade que surgiram logo a seguir.

Em 1972 ocorreu a fundação da Atari, de Nolan Bushnell e Ted Dabney. Que lançou o fenômeno ***Pong*** para arcade no mesmo ano, jogo desenvolvido por Allan Alcorn.
Em 1974 o mercado havia saturado de jogos de bola e raquete, o que causou uma queda nas vendas de jogos. Então as empresas começaram a lançar jogos de corrida, luta e tiro. Alguns exemplos foram ***Tank***, ***Gunfight***, ***Wheels***, e ***Sea and Wolf***.
Em 1978 que então começa “a era de ouro” dos videogames. Neste período os jogos de videogame estavam bem estabelecidos, mas sua popularidade ainda era inferior ao pinball, por exemplo.
Mas foi necessário apenas um jogo para mudar isto: ***Space Invaders***, para arcade.

Este jogo introduziu diversos conceitos populares, como vidas ao invés de um tempo, vidas extras que dependiam da pontuação, e poder ver as pontuações de outros jogadores em um ranking interno na máquina. Também foi o primeiro jogo a incluir ondas de inimigos que atacavam o jogador, e a incluir músicas de fundo.
Devido ao fenômeno que se tornou, vendeu mais de 200 mil unidades rapidamente somente no Japão. Nos Estados Unidos, foi um sucesso que vendeu mais de 60 mil unidades.

Em 1980, lança para arcade ***Pac Man***, que vende quase 100 mil unidades apenas nos Estados Unidos, o que inicia uma nova era de personagens identificáveis e com foco. Outro jogo que também era focado nos personagens e foi lançado nesta época foi ***Donkey Kong*** em 1981, e o primeiro jogo da franquia Mario.

Em termos econômicos, a indústria de arcades subiu de 300 milhões de dólares em 1978, para 900 milhões de dólares em 1979, e para 2,8 bilhões de dólares em 1980.

Graças a ***Pac Man*** e outros jogos com personagens com foco, a indústria passou a circular 5 bilhões de dólares em 1981, e 8 bilhões de dólares em 1982.
Com estes valores, a indústria de jogos havia então superado ambas as indústrias de música pop (4 bilhões de dólares) e a de Hollywood (3 bilhões de dólares).

Hoje, a indústria de jogos conta com receitas que devem chegar a 187,7 bilhões de dólares em 2023 (Laurence, 2023, p.1), e foi produzido o filme do jogo Super Mario Bros. do qual conta com um elenco de atores de Hollywood e conseguiu atingir o segundo lugar nas bilheterias mundiais com a quantia de 1,3 bilhões de dólares.(Assumpção, 2023, p.20)

2.1.3 Sobre empresas de jogos

Umas das mais conhecidas e populares empresas de jogos no começo foram Nintendo e Sega, no começo de 1983 começa a aparecer no Japão a terceira geração de consoles, sendo dois importantes lançamentos o Famicom (*Nintendo's Family Computer*) e o SG-1000 da Sega.
O sucesso do Famicom foi muito maior e ocasionou o lançamento rápido do *Sega Mark III* em 1985.
Para fazer sucesso nos Estado Unidos, a Nintendo alterou o design completamente e o chamou de NES(*Nintendo Entertainment System*) e o lançando somente em 1985.
Ao fim de 1989, o mercado de jogos baseados em cartuchos era de mais de 2 bilhões de dólares, enquanto que o de discos era de apenas 300 milhões, o que fez várias publicadoras como a Eletronic Arts e a LucasArts ficarem muito mais em jogos de console.
Arcades fizeram um ressurgimento em 1991, quando a Capcom lançou ***Street Fighter II***, popularizando os jogos de luta, revivendo os arcades e influenciando jogos como ***Mortal Kombat*** e ***King of Fighters***.
Um jogo que se destaca nesta época devido ao seu orçamento era ***Final Fantasy VII*** apenas para Playstation, da Squaresoft, jogo que foi criado exclusivamente para os competidores que usavam CDs.

Em 2000, a Sony lança o Playstation 2, que possibilitava a leitura de DVDs, e possuía um processador e gráficos melhores quando comparado ao Playstation. Também era possível assistir CDs e DVDs no console.

2.1.4 Tipos de jogos

Na década de 90 houve muitas inovações. Foi nesta época que ocorreu a transição para gráficos 3D, e vários novos tipos de jogos começaram a aparecer, como tiro em primeira pessoa, estratégia em tempo real, e MMORPGs (RPG Online Massivo).
Os jogos podem variar muito de gênero, o tipo de foco e a jogabilidade ou design.
A Sega e a Nintendo, por exemplo, tinham ideias diferentes para seus jogos. Enquanto a Sega focava em jogos que eram inovativos e únicos em termos de tecnologia e jogabilidade, a Nintendo tentava estabelecer franquias longas e populares, mas sem muitas inovações entre um jogo e o outro da mesma série.
Muitas franquias de sucesso que existem até hoje, e que são bem diferentes não só no gênero mas também na forma como afetam o jogador, tiveram início na mesma época. Exemplos são: ***The Legend of Zelda***, ***Dragon Quest***, ***Phantasy Star***, ***Metal Gear***, e ***Final Fantasy*** (que salvou a Square da falência).
Todos estes jogos tem elementos semelhantes, porém têm aproximações diferentes sobre como são jogados e evoluíram de maneiras diferentes.
Por exemplo, todos têm uma narrativa envolvente, personagens envolventes e alguns foram mudando de gênero e aproximação com o tempo, mas sem perder a essência.

2.1.5 Sobre “Mecanismos de Jogos”

Os Mecanismos de jogos (*Game Engines*) foram criados para diminuir o trabalho necessário para desenvolver um jogo, facilitando o trabalho de ter que criar toda uma base de programação e funcionalidade.
Com ferramentas como essas, uma pessoa seria capaz de desenvolver um jogo sozinha, sem ter que precisar de toda uma equipe. Já que a *Game Engine* disponibiliza diversas ferramentas para desenvolver jogos em adição de *softwares* reutilizáveis.

2.2 Game Engines

2.2.1 O que são Game Engines

O termo "*Game engine*" surgiu em meados dos anos 90, especialmente em conexão com jogos 3D como por exemplo FPS (*First Person Shooters*).
Em suma, *Game Engines* são estruturas de software projetadas primariamente para o desenvolvimento de jogos e geralmente inclui bibliotecas relevantes e programas suportes.
As *Game Engines* comumente conhecidas são *softwares* de desenvolvimento que utilizam esta estrutura, tipicamente oferecendo um conjunto de ferramentas e características para o desenvolvimento de jogos.

Estas ferramentas geralmente são providas em um ambiente de desenvolvimento integrado para permitir o desenvolvimento de jogos rápido baseado em dados de maneira simplificada.

Segundo um post do Blog da PixStudios(2014, p.8),

> "Para desenvolver uma *Game Engine* seria muito caro e trabalhoso, tendo que haver a necessidade de dezenas de programadores escrevendo milhares de linhas em *C#*, *Java* e *Python*, para que um motor de jogos seja criado. Hoje, a maioria dos que existem no mercado, foram feitos para um título específico e depois foram liberados para o público em geral. Os preços variam muito, alguns podem ser gratuitos, já outros podem custar até mesmo mais de 30 mil dólares! Porém, muitas delas têm um plano gratuito para estudantes ou jogos não-comerciais."

Antes das *Game Engines*, os jogos eram escritos tipicamente como entidades singulares: um jogo para o Atari 2600, por exemplo, tinha que ser projetado do zero para fazer uso otimizado do hardware de exibição. Havia restrições de memória que acabavam sabotando tentativas de criar *designs* de dados pesados que uma *engine* precisava.
O código teria que ser jogado fora depois de qualquer forma, já que gerações futuras de jogos usariam designs completamente diferentes que tomariam vantagem de recursos extras.
Desde a época de ouro dos jogos, se tornou comum para as empresas de jogos desenvolverem *Game Engines* próprias para uso de seus softwares originais.

2.2.2 Exemplos de Game Engines

Unity 3D: É uma das *engines* mais utilizadas do mundo. Ela possibilita a criação de jogos 2D e 3D com otimização alta e gráficos impressionantes. Utiliza a linguagem C# e oferece a possibilidade de escrever as linhas de código, garantindo mais liberdade na hora de desenvolver.
Suas características principais são: suporte para o uso de *shaders*; suporte ao *PhysX*, incluindo detector de colisão, *soft body* e *ragdoll*; compatibilidade com os navegadores (via o plugin ***Unity Web Player***); compatibilidade com ***Blender***, ***3ds Max***, ***Maya***, ***Cinema 4D***, ***Cheetah 3D***, ***Softimage***, ***modo***, ***ZBrush***, ***Lightwave***, ***Photoshop***, ***Fireworks***, e "***Substance***".
A Unity é notável por sua capacidade para múltiplas plataformas. Os desenvolvedores têm controle sobre a criação dos jogos para dispositivos móveis, *web browsers*, *desktops* e *consoles*.
Alguns jogos populares feitos na Unity: ***Among Us***, ***Cuphead***, ***Fall Guys: Ultimate Knockout***, ***Garena Free Fire***, ***Genshin Impact***, ***Hollow Knight***, ***Pokémon GO***, etc.

Unreal Engine: A Unreal é muito poderosa e tem seu foco na realidade gráfica. A programação pode ser feita em C++ ou usando ***Blueprints***, uma programação visual de arrasta e solta mais avançada.
Alguns jogos mais conhecidos da Unreal: ***Batman: Arkham City***, ***Bioshock Infinite***, ***Borderlands 2***, ***Injustice: Gods Among Us***, ***Life is Strange***, ***Mortal Kombat 11***, ***Fortnite***, ***Kingdom Heart 3***, etc.

Godot Engine: É completamente gratuita e *open-source*. Permite criar jogos em 2D e 3D usando uma arquitetura baseada em nós e um editor visual. O código pode ser escrito nas linguagens ***GDScript*** (baseada em Python), ***C++***, ***C#*** e outras com suporte adicionado pela comunidade.
Alguns jogos da *engine*: ***Sonic Colors: Ultimate*** e ***Dome Keeper***.

Game Maker: É considerada por muitos, a melhor *engine* para jogos 2D. Conhecida pela sua facilidade de uso. Possibilita criar jogos sem programar, apenas arrastando

e soltando componentes, mas também possibilita escrever as linhas de código usando ***Game Maker Language*** (GML).
Alguns jogos feitos na *engine*: ***Katana Zero***, ***Undertale***, ***Hotline Miami***, ***Forager***, ***Cats Organized Neatly***, etc.

Construct: *Engine* voltada para criação de jogos 2D que rodam direto no navegador. Também permite criar jogos sem programação, usando uma interface arrasta e solta. É baseada em HTML5, e faz o uso de lógica e programação visual, orientada a condições e eventos.
Alguns jogos pela *engine*: ***6Souls***, ***8BitBoy***, ***Bad Time Simulator***, ***Deep Space Rush***, etc.

Phaser: É uma estrutura usada para jogos 2D, em HTML5, via *mobile* ou desktop. O desenvolvimento pode ser feito em HTML e também em ***JavaScript***. Essa estrutura usa renderizadores ***Canvas*** e ***WebGL***.
Jogo exemplo feito pela *engine*: ***Vampire Survivors***.

RPG Maker: exclusivamente voltado para a criação de RPGs 2D, no estilo 16 bits. A *engine* não exige a escrita de linhas de código, sendo no estilo arrasta e solta.
Jogos notáveis da *engine*: ***Omori***, ***Your Turn to Die***, ***OneShot***, ***Mad Father***, ***Ao Oni***, ***Ib***, ***The Witch's House***, etc.

CryEngine: Foi criada pela Crytek e é especializada em jogos FPS. Possui um poderoso editor visual, diversas ferramentas integradas e de código aberto.
Exemplos de jogos da *engine*: Série ***Far Cry***, Série ***Crysis***, ***Prey***, ***Sniper: Ghost Warrior 3***, etc.

2.2.3 Diferenças entre *Game Engines*

Desenvolvedores podem usar as *Game Engines* para construir jogos para consoles e outros tipos de computadores.
As funcionalidades essenciais tipicamente providas podem incluir um renderizador para gráficos 2D ou 3D (ou ambos), motor de física ou detector de colisão (e resposta de colisão), som, *scripting*, animação. inteligência artificial, rede, transmissão, gerenciador de memória, *threading*, suporte de localização, gráfico de cenário e suporte de vídeo para cinemáticas.
As *Game Engines* podem se diferenciar no foco que possuem, com o tipo de jogo e a plataforma alvo. (Ver exemplos de *Game Engines* para detalhes)

2.2.4 Sobre a Game Engine utilizada neste estudo

A *Game Engine* Unity foi escolhida pela preferência no uso da ferramenta, pelos métodos de ensino e ferramentas fáceis de usar e de alta compatibilidade.
Não existe melhor ou pior *engine* para criar jogos ou ambientes e cenários/objetos 3D/2D, é como escolher a linguagem de programação que o usuário irá aprender.
Neste estudo são apresentados os pontos positivos sobre a Unity, os métodos de ensino básico e sobre as ferramentas principais. Ao final todos os conceitos e teorias aprendidas sobre Unity serão praticadas em um estudo de caso, um jogo em fase exploratória desenvolvido ao longo do estudo.

2.3 Pontos da Unity

2.3.1 Valores da Unity, como convivem com eles e seus números

Tópico sobre os valores que a Unity defende e busca ser. Informando como trabalham e tratam uns aos outros diariamente e como ajudam a tomar as decisões certas para os clientes, parceiros e colaboradores.

Eles colocam os usuários em primeiro lugar, como se fossem estrelas e eles (Unity) seus fãs.

Essa dedicação que compartilham com o usuário os une, define e alinha o trabalho e os orienta a atendê-los.

Acreditam que ideias possam surgir de qualquer lugar. Então realizam debates vigorosos, escutam e aprendem com o intuito de garantir que a melhor ideia prevaleça. Se importam o suficiente para passar pela dor de conversas confusas.

Respeitam a individualidade de cada pessoa e estão juntos nessa, garantem que tenha voz e que seja ouvida.

Arriscam, e com a falha aprendem para melhorarem e arriscarem novamente. Se desafiam a se elevarem uns aos outros além do limite para fazer além do impossível. Se mantém curiosos e famintos.

A empatia é essencial para validar a perspectiva das outras pessoas. A capacidade de sentir o mesmo que o outro e se colocar no lugar dela.

O respeito é conhecer a dignidade. Tratar o próximo da maneira como ele deseja ser tratado.

Oportunidade é garantir que todas as pessoas sejam tratadas igualmente, livres de preconceitos, barreiras artificiais ou preferências.

Acreditam que os criadores fazem do mundo um lugar melhor. Isso está no núcleo do negócio deles, acreditam que sua tecnologia é capaz de mudar o mundo.
Seus produtos oferecem ferramentas que são capazes de entreter e criar experiências RT3D inovadoras e fornecer processos melhores para quase todos os setores.

Em 2021, a Unity fez o levantamento de seus dados e apresentou as seguintes informações:

- 5 Bilhões de *downloads* por mês de aplicativos desenvolvidos com Unity.
- 70% dos 1000 principais jogos para dispositivos móveis foram feitos com Unity.

- 50%+ dos jogos para dispositivos móveis, PC e console foram feitos com Unity.
- 3.8 Bilhões de usuários ativos mensais que consumiram conteúdo criado ou operado com soluções da Unity.
- 20+ diferentes plataformas executam as criações do Unity.
- 190+ países e territórios tem criadores do Unity.

2.3.2 Sobre a Unity

Unity é a criadora da ferramenta de RT3D que lidera no mundo todo, dando aos usuários as ferramentas mais poderosas e acessíveis para criar, operar e monetizar experiências para o mundo em tempo real.
A Unity empodera qualquer um, independente da habilidade ou indústria, a criar conteúdo visual 3D usando tecnologia de classe mundial, operando usando recursos que maximizam a facilidade de usar, e monetizar, para que encontrem sucesso com suas criações.
O time de 1000 pessoas na equipe de desenvolvimento deixam a Unity na frente do desenvolvimento ao trabalhar lado a lado com parceiros como ***Google***, ***Facebook***, ***Oculus***, ***Autodesk*** e ***Microsoft*** para garantir suporte otimizado aos lançamentos recentes e plataformas.

Unity cria e desenvolve várias ferramentas que podem auxiliar em outros pontos além do desenvolvimento de jogos, como por exemplo, um Simulador de jogos que, através do poder da simulação em nuvem, ajuda criadores a alcançarem o balanceamento de jogo ideal.

Unity Game Simulation reduz significativamente o tempo e o custo associado ao teste de pré-lançamento simulando milhares de jogadas rapidamente e precisamente na nuvem, ajudando os estúdios de jogos de todos os tamanhos e orçamentos a alcançarem o balanceamento desejado.
Ao lançar no ***Google Cloud***, Unity é capaz de operar a ***Unity Game Simulation*** em escala, resultando em estabilidade mais alta, segurança elevada, e contínua integração e entrega.

Unity Games Simulation é designada a complementar o método tradicional de balanceamento de jogos e aliviar testadores da pesada, repetitiva característica de testar. A tecnologia também serve como alternativa para ferramentas imprecisas e incômodas, como planilhas, usadas atualmente em estúdios de hoje. Com a ***Unity Games Simulation***, criadores de jogos podem passar mais tempo focados no aspecto centro-humano de criação de jogos, como design, enquanto realoca orçamento do jogo para iniciativas que não são testes como marketing e construção de comunidade. Isso, em troca, resulta em tempo de lançamento mais rápido e percepção do jogador geral aumentada sobre jogos.

De acordo com alguns sites de notícias, um relatório de Silver Screen Beat disse que a Unity está utilizando sua engine para desenvolver humanos realistas virtuais chamados "*No Player 's Characteristics*" ou NPCs.(Morales, 2022, p.1)
Estes podem ser usados para testar tudo desde a segurança das montanha-russas até situações do tráfego do mundo real para um veículo automotivo.
De acordo com com o supervisor da Unity, veterano vice-presidente de IA Danny Lange, pode-se recriar um mundo que é melhor que o da realidade para treinar sistemas num mundo sintético.
Ele adiciona que pode criar muitos outros cenários "usando dados na Unity". Por exemplo, isto pode algum dia ajudar a fazer veículos automotivos uma realidade.

No evento "***Unity For Humanity***" de 2022, foram destacados os vencedores que utilizaram a ferramenta para mudar o mundo de alguém para melhor.
Este programa foi criado para auxiliar criadores que querem "fazer do mundo um lugar melhor", os vencedores recebem financiamento e mentoria para seus projetos.(Rousseau, 2022, p.1)
Dentre os premiados estão pessoas que utilizaram a música, a realidade aumentada, realidade virtual, mobile, arte digital e mídia mista. Muitos destes projetos foram criados para aliviar a dor e os traumas de muitas pessoas.

A Unity Software adquiriu ***Ziva Dynamics*** em 2022, uma companhia de tecnologia *machine learning* que provém simulação e modificação machine learnings e criador de personagem em tempo real.

Em uma postagem em seu blog, Unity diz que Ziva está resolvendo o problema de criar “humanos digitais” que atendem a qualidade e dinâmicas naturais necessárias.
A tecnologia da Ziva já foi utilizada nas indústrias de jogos, filmes e televisão, em títulos como ***Game of Thrones***, ***Godzilla vs. Kong***, e ***Senua’s Saga: Hellblade II***.(Boughedda, 2022, p.1)

Com essa tecnologia, Unity lança uma demonstração tecnológica chamada “***Enemies***”, que demonstra o que ela é capaz de fazer graças a ela.(Peters, 2022, p.1)
O vídeo mostra uma mulher numa sala surpreendentemente realista, ela é renderizada em detalhes impressionantes, seu cabelo muda de maneira sutil enquanto ela se move, seus olhos olham ao redor da sala e então ao bispo do xadrez de uma maneira super natural, e o close final filma seu rosto que parece como se fosse mesmo o de um humano real.

Ziva Dynamics desenvolveu a ferramenta Ziva RT na esperança de simplificar o processo de criação de personagens digitais realistas para artistas de todos os níveis.
Com uma personagem já bem realista, mas ao invés de tomar um longo tempo para ter cada quadro de animação processado por um computador incrivelmente poderoso, Emma (uma “humana virtual”) é capaz de rodar em tempo-real, com mudanças feitas na animação sendo refletidas quase que instantaneamente.
Isto é significante pois pode demorar meses para uma pequena equipe criar e animar um personagem com a qualidade média para um jogo. Para animações realistas que vemos em filmes e jogos como na franquia ***The Last of Us***, estúdios usam capturas de movimentos para as performances, mas acabam com clipes de animações que não podem reagir dinamicamente.
Em comparação, Ziva RT tem o potencial de permitir criadores a pegarem o modelo 3D, fazer upload para processamento na nuvem, e ter de retorno um manequim da face dentro de uma hora que pode equiparar-se à qualidade visual de face de capturas de movimento.
Crucialmente, estes personagens podem ter suas animações ajustadas dinamicamente, fazendo deles idealmente adequadas para jogos.

O objetivo desta tecnologia é de fazer “criação de personagem realista acessível e escalável para todos os artistas independente do nível de habilidade,” diz Tatarchuk.

2.3.3 Utilizações da Unity

Unity pode ser utilizada em modelagem para propósitos que não sejam sobre jogos, várias empresas utilizam a ferramenta para este propósito.

Uma tecnologia, que cria modelos virtuais de cidades e simula cenários políticos, poderia ser uma potencial mudança de jogo no planejamento urbano.
Orlando, Flórida, foi transformada em espaço virtual pela ***Orlando Economic Partnership*** (OEP) graças a uma parceria feita com a Unity. O modelo 3D da cidade pode mostrar a potenciais investidores a sua oferta para crescer como um centro de tecnologia.
Como uma *SimCity* para *policymakers*, “***digital twins***” permite às cidades não somente criarem modelos virtuais, mas a executar simulações de novas políticas ou projetos de infraestrutura e pré-visualizar seus potenciais impactos antes de tomar uma decisão no mundo real.
A tecnologia poderia ajudar oficiais a cortar custos e emissão de carbono de novas construções, e evitar modificações custosas depois de um projeto ser finalizado.
Em meio a uma crise climática cada vez mais iminente nas áreas urbanas, poderia habilitar as cidades a testar a efetividade de várias medidas contra os níveis elevados do mar e calor urbano. Com uma estimativa, ***digital twins*** pode economizar para cidades uns $280 bilhões até 2030.(Poon, 2022, p.6)

Enquanto a maioria das cidades permanecem na estágios iniciais da construção dos próprios ***digital twins***, Singapura virtual já está como uma amostra para o que a tecnologia é capaz.
O modelo da nação-ilha comprime mais de 3 milhões de imagens capturadas ao nível da rua e 160,000 imagens tomadas do céu, junto com bilhões de pontos de dados plotados em 3D, totalizando mais de 100 *terabytes* de puros dados. A fundação do modelo contará com 14 núcleos de pontos de dados em tudo, desde o uso do terreno a cobertura de árvores a utilidades do subsolo.

O modelo é considerado chave no objetivo da cidade-nação de construir sustentabilidade.
Com aproximadamente 6 milhões de pessoas morando em apenas 280 milhas quadradas de terra, há pouco espaço para erro.(Poon, 2022, p.9)

Com inovação, tecnologia e aventura no seu núcleo, a empresa de design de barcos ***Arksen*** precisava de um novo jeito de demonstrar seu mais recente recipiente de exploração marítima que alinhava se com esses valores fundamentais.
Quando eles se associaram com uma empresa de design especializada em criar espaços 3D compartilhados que trabalhavam entre área de trabalho, mobile, realidade aumentada e virtual, ***canVERSE*** sabia exatamente onde começar.
canVERSE alavancou ***Unity Forma*** a criar um fluxo de trabalho simplificado, começando com os dados do design auxiliado pelo computador da ***Arksen*** (*computer aided design*, CAD) e resultando em um configurador interativo RT3D do navio mais recente deles.
Eleanor Briggs, diretor de marketing da ***Arksen***, diz: “Antes, criar uma renderização estática de nossos navios levaria algumas semanas para coordenar.
“Agora, o modelo parece estar vivo já que os clientes podem interagir com a ***Arksen 85*** em tempo-real, vendo de qualquer ângulo enquanto customizam opções com ***Unity Forma***.”(Edwards, 2022. p.7)

2.4 Metodologia da Unity

A seguir será explicado como é a metodologia de aprendizagem da Unity de acordo com a experiência no uso da ferramenta. Será mostrado como pode ser mais simples do que se imagina o que precisa para começar a desenvolver um jogo e será explicado exemplos de uma sessão de aprendizagem.

2.4.1 Como a Unity ensina?

Unity disponibiliza ensinos online de maneira gratuita por seu site. Ela permite que o usuário comece por um curso que foi criado por eles, criadores da Unity, para desenvolvedores que querem utilizar Unity.
Seus caminhos estruturados por eles mesmos levam o auto-aprendizado para o próximo nível.

Foram projetados de maneira hábil para otimizar sua experiência de aprendizado, ensinando-o exatamente o que o usuário tem que saber, quando precisa saber, com um currículo passo-a-passo.
Cada caminho contém missões discretas. Se já estiver familiarizado com o conteúdo, o usuário pode testar suas habilidade para pular cada ponto de controle.
Ganha-se pontos de experiência ao completar cada tutorial, quanto mais pontos, mais próximo de se tornar um Profissional Unity.
Existe toda uma comunidade de companheiros que estão aprendendo Unity e que podem compartilhar progresso e motivação. O usuário pode fazer perguntas, tirar dúvidas e se juntar para poder falar com a equipe ***Unity Learn***.
Para cada caminho que o usuário concluir, ganhará uma credencial digital compartilhável.
Ao completar pontos de controle, libera-se alguns pacotes de *assets* da Unity que podem ser usados em seus projetos.

2.4.2 Caminhos da Unity

Neste curso, a Unity recomenda iniciar pelo caminho Essenciais da Unity, feito para iniciantes da Unity, guiando durante seus primeiros passos através do ganho de conhecimento sobre contexto e habilidades para criar no Editor da Unity e trazer suas visões à vida.
A Unity também encoraja a publicar seus projetos e ver os projetos de outros desenvolvedores, além de desafios extras e acesso a uma sessão de comentários para tirar dúvidas com outros usuários.
A Unity proporciona o usuário a realizar a experiência na prática, e ao completar este caminho, o usuário estará equipado com a fundação necessária para aprofundar seu aprendizado e se especializar na área de interesse.
Depois de completar o Essenciais da Unity, o usuário tem a opção de ir para os caminhos: Programador Júnior, Núcleo Criativo e Desenvolvimento AR (*Augmented Reality* ou Realidade Aumentada) para *Mobile*.

Sobre o Programador Júnior, este caminho guia levará o usuário do zero até estar pronto para um trabalho!

O caminho é para interessados em código ou quem quer obter um nível de entrada de função na Unity, o caminho assume que o usuário tem conhecimento básico sobre a ferramenta e não tem pré-requisitos matemáticos. Além disso, a Unity emite um Certificado Unity para certificar das habilidades aprendidas durante o desenvolvimento dos trabalhos.

Após completar todo o caminho Programador Júnior o usuário tem acesso ao caminho Desenvolvimento VR(*Virtual Reality* ou Realidade Virtual). Este caminho prepara o usuário para um trabalho na indústria de VR.
Ele é projetado para qualquer um que esteja interessado em aprender a criar experiências para o VR. Assume-se que o usuário tenha conhecimento básico da Unity e de programação.

No caminho de Desenvolvimento AR para Mobile o usuário criará aplicativos AR compatíveis com dispositivos iOS e *Android*!
O usuário utilizará o *Visual Scripting* e as ferramentas da Unity para AR para desenvolver uma variedade de experiências engajadoras AR. O caminho assume que o usuário saiba do básico da Unity.

Com o caminho Núcleo Criativo, melhora-se os entendimentos essenciais da Unity com aspectos criativos da ferramenta.
É o próximo passo para o usuário tornar-se um criador Unity. Assim que o usuário assimilar todos os ensinamentos do Essenciais da Unity, neste caminho aprenderá sobre VFX(*Visual Effects* ou Efeitos Visuais), Iluminação, Animação, Audio, UI(*User Interface* ou Interface de Usuário) e outras habilidades criativas, sem precisar de programação.

Fora todos estes caminhos criados pelos Desenvolvedores da Unity, há também várias redes sociais para acessar e aprender com outros alunos e professores. Há vários outros cursos específicos e várias transmissões ao vivo sobre diversos assuntos.

Todo o aprendizado pode não estar 100% localizado em português, mas com tantas opções extras, como as redes sociais e *livestreams*, sempre pode haver pessoas de todas as línguas dispostas a ensinarem quem busca conhecimento e quer melhorar.

2.4.3 Afundo da Unity *Essentials*

Como o caminho ***Unity Essentials*** é tudo sobre aprender a usar a ferramenta Unity, é melhor separar uma parte especialmente para explicar sobre o uso da ferramenta. Esta parte focará em explicar sobre este caminho resumidamente.

Durante todo o caminho, são apresentadas informações de criadores Unity estabelecidos sobre as jornadas deles, e irá explorar uma variedade de recursos disponíveis para ajudar no seu aprendizado.

O objetivo dos projetos é aprender sobre: explorar as razões de querer aprender sobre Unity, explorar a mentalidade e habilidades requeridas para aprender Unity, identificar os principais desafios e oportunidades na jornada de aprendizado Unity, identificar os elementos chave do Ecossistema de aprendizado Unity e seus propósitos.

Depois de instalar os programas da Unity, criar uma conta e escolher um plano de uso da ferramenta, haverá a opção de abrir um *Microgame*, que são jogos feitos para aprender mais sobre a ferramenta e sua interface, estes jogos vem com tutoriais embutidos e te ajudam a ficar familiarizado com todas as opções da ferramenta. Também existe a opção de pular a parte de *Microgames* e ir para a parte de criar projetos, ouvir dicas de criadores do mundo todo e completar desafios.
Caso o desenvolvedor for familiarizado com a ferramenta, pode realizar um “teste” para sair deste caminho ao completar as missões de ponto de controle.
Ao completar ou pular essas etapas, será recomendado alguns recursos como começar com um dos kits dos criadores (RPG, FPS, ou *puzzle*), mexer com códigos, optar para o caminho do programador júnior, explorar os eventos de transmissão ao vivo, e explorar centenas de outros recursos de aprendizado.

Continuando no caminho de essenciais da Unity, na parte de criar projetos o usuário ouvirá muito sobre os criadores e verá como usam Unity até hoje. Todos começaram da mesma maneira!
Os criadores irão compartilhar sobre as felicidades e frustrações de aprender Unity, e o porquê de valer a pena. A ferramenta é bastante flexível, o que a torna bastante agradável aos criadores.
Ao ver alguns vídeos dos criadores, pense sobre o que antes não era possível, mas agora será possível com a Unity, como será seu jeito favorito de se expressar na Unity? Pense em mais perguntas que irão lhe ajudar a saber o que fazer com Unity.
Os criadores compartilham também sobre como decidiam o que criar, enquanto eram realistas sobre as habilidades e recursos deles na época. O importante é sempre começar simples, com os básicos, e se motivar por aquela ideia especial - a ideia que lhe motivará a seguir em frente para aprender mais.

Quando se trata de aprender Unity, acredita-se que os criadores precisam do *mindset* certo. Um *mindset* pode ser descrito como aquelas crenças que o guiam quando o usuário está resolvendo problemas ou lidando com situações. Os criadores compartilham suas recomendações que devem trazer o *mindset* para aprender Unity.
Os *mindsets* mencionados são: curiosidade, solução de problemas, motivação para aprender, comunicação, colaboração, auto-estima, paciência, foco, positividade e persistência.
Quais destes *mindsets* o usuário acha que possui? Quais o usuário acredita que precisa melhorar? Considere estes *mindsets* na sua jornada criativa e o usuário pode acabar cultivando-as, as *mindsets* que ajudaram os criadores estabelecidos a chegarem onde estão!
E sobre as habilidades necessárias para o sucesso na Unity? Mesmo que alguns criadores tenham dito habilidades em arte ou programação, o consenso é que tudo que precisas é de um *mindset* positivo e acreditar que conseguirás! Na verdade, muitos criadores Unity já disseram, “se aprendi Unity, qualquer um aprende!”

Muitos desenvolvedores *indies* são grupos de poucas pessoas, havendo grupos desde as dezenas até os grupos de 3 pessoas, e muitos deles tem pessoas que

focam em seus pontos fortes, seja a programação ou direção de arte, composição de trilhas sonoras ou *marketing*.
Em alguns casos, o desenvolvedor *indie* é capaz de realizar vários trabalhos e funções, porém a quantidade de aprendizado e esforço deve ser proporcional ao conhecimento dele.
Alguém que está começando sua jornada de aprendizado deve ter em mente quais habilidades ela possui, quais ela irá desenvolver e quais ela deixará encarregados para outros.

Ao aprender o básico e prosseguir para a próxima parte, o usuário irá explorar as funções e caminhos de carreiras de um criador Unity. Aprenderá sobre os usos da Unity (que são mais do que jogos!), quem cria, e que posições eles têm.
Os tópicos falam sobre criações em tempo real, os essenciais sobre 3D em tempo real, essenciais de programação da Unity, essenciais de áudio em tempo real, essenciais sobre 2D em tempo real, e a indústria em tempo real.

No final do ***Unity Essentials***, o usuário irá identificar e planejar sua própria jornada de aprendizado, ao criar um plano de ação de aprendizado para seu próximo caminho na aprendizagem da Unity.

3. Estudo de Caso do Jogo

Uma breve descrição sobre o jogo desenvolvido durante a realização deste estudo é apresentada neste capítulo. São apresentados os detalhes de cada parte do desenvolvimento durante a implementação. Nota-se que esta parte é necessária o conhecimento de programação, mais especificamente sobre C#. Será dado uma breve explicação das funcionalidade utilizadas.

O jogo é em visão 2D, o personagem pode ser controlado para a esquerda e direita, pode pular uma vez enquanto estiver no chão, pode atacar para a esquerda e direita, e o tempo todo a câmera foca no centro da tela com inimigos surgindo dos lados e de cima.
O objetivo é sobreviver o máximo que conseguir enquanto derrota os inimigos que vêm em sua direção.
A próxima parte será inteiramente focada em falar detalhes sobre os aprendizados iniciais que tornam possível criar o jogo.

Para melhor compreensão sobre a explicação do desenvolvimento do jogo será apresentado os termos usados, mecânicas, funções e um pouco da linguagem de programação utilizada.

3.1 Termos da ferramenta

Inspector

O inspetor mostra as propriedades do *GameObject*, *script* ou *asset* selecionado.
Pode-se travar um inspetor específico para focar nas propriedades de algo que será editado frequentemente, por exemplo, quando se está ajustando o posicionamento da câmera e é possível deixar aberto um segundo inspetor para outros usos.

Materiais & *Shaders*

Para ter algo na Unity, é preciso prover informação que descreve o formato, e informação que descreve a aparência na superfície. É usado malhas para descrever os formatos, e materiais para a aparência.
Materiais e *Shaders* estão linkados, sempre é usado materiais com *shaders*.

O material contém uma referência ao *shader*, que pode definir as propriedades do material, então ele pode conter dados sobre cores e referências a texturas.

Componentes

Componentes são usados para adicionar funções aos *GameObjects*, essas funções podem ser controladas pelo inspetor ou usando um *script*. Eles podem ser adicionados pelo menu de componentes para, por exemplo, aplicar propriedades físicas em um objeto usando o componente *RigidBody*. Existem diversos componentes que podem ser usados como Áudio, Efeito, Eventos, *Mesh*, *Render*, etc.

GameObjects

O *GameObject* é o conceito mais importante do editor Unit. Todo objeto é um *GameObject*, desde personagens e coletáveis até a luz, câmeras e efeitos especiais. Porém, os objetos não fazem nada por si só, eles precisam de propriedades e componentes para se tornarem o que queremos. A única componente que vem e não é possível tirar é a *Transform*, que representa a posição e orientação do objeto.

Prefabs

O sistema de *Prefab* permite criar, configurar e guardar um *GameObject* completo com todos seus componentes, propriedades e objetos-filhos como *Asset* reusável. O *Prefab Asset* atua como um modelo do qual o usuário pode criar novas instâncias de *Prefabs* no cenário.

Isto é para quando se quer usar novamente um *GameObject* que está configurado de uma maneira específica ou particular, como um NPC(*Non-Player Characte*r ou Personagem não jogável) ou um pedaço do cenário, em múltiplos locais e em múltiplos cenários diferentes.

Desta maneira não será preciso ficar copiando e colando o *GameObject*, pois o sistema de *Prefab* permite que todas as cópias sejam deixadas em sincronia. Qualquer mudança que é feita em um *Prefab Asset* é automaticamente refletida nas outras instâncias daquele *Prefab*, permitindo que sejam feitas mudanças pelo projeto todo sem haver que fazer repetidas edições.

Nem todos os *Prefabs* têm que ser idênticos, pode-se salvar configurações individuais em certos *Prefabs* para que sejam diferentes dos outros.

RigidBody

RigidBody permite aplicar comportamentos da física, como gravidade, massa, resistência e *momentum*.

Collision

Para configurar colisões na Unity, é preciso usar o componente *Collision*, que define o formato do *GameObject* para propósitos de colisões físicas. Pode-se usar estas colisões para gerenciar eventos baseados em colisão. Por exemplo, para saber quando dois objetos colidiram e ativar algum evento em resposta a isso.

Assets

Um *Asset* é qualquer item que é usado no projeto Unity para criar um jogo ou app. *Assets* podem representar um elemento visual ou um áudio, como modelos 3D, texturas, atos, efeitos sonoros, ou música. Podem ser representados por itens mais abstratos como gradação de cores, marcas de animação, ou texto arbitrário ou data numérica para qualquer uso.

3.2 Termos do C#

Scripts

Um pedaço de código que permite criar componentes próprios, gatilhos de eventos do jogo, modificar propriedades dos componentes ao longo do tempo e responder a *input* do usuário da maneira desejada.

Input

Input permite ao usuário controlar a aplicação usando um dispositivo, toque, ou gestos. Pode-se programar elementos dentro do *app*, como a interface de usuário, para responder ao input do usuário de maneiras diferentes.

A Unity suporta input de diversos tipos de dispositivos, incluindo: mouse e teclado, analógicos, controles, *touch screens*, controles VR e AR.

O gerenciador de *Input* permite definir eixos de entrada e suas associadas ações para seu projeto.

GetKey

Pode-se pedir por um *input* específico ou botão com *Input.GetKey* e o nome da "tecla" desejada, por exemplo, *Input.GetKey("Space");* para ler a entrada da tecla "Espaço" e realizar a ação que quiser.

AddForce & ForceMode

São parte do motor de física e são atualizados em tempos discretos de forma similar a maneira que os quadros renderizam.

Basicamente são usados para adicionar e regular movimento aos objetos, por exemplo, para fazer um personagem pular é usado o *AddForce* para "empurrar" o objeto para "cima" e o *ForceMode* para ajustar a intensidade do "empurrão".

Vector3

Representação de vetores e pontos 3D. Esta estrutura é usada por toda a Unity para passar posições e direções 3D por aí. Possui também funções para fazer operações vetoriais comuns.

LookDirection

Código usado para fazer um objeto ir em direção a uma coordenada especificada. Por exemplo, fazer um "inimigo" perseguir o "jogador".

Destroy

Código para destruir um objeto especificado, por exemplo, para destruir um objeto que "saiu" do cenário ou um "inimigo" que foi "derrotado" pelo "jogador".

Random.range

Código que escolhe um alcance aleatório dentre números à coordenadas. Por exemplo, fazer um "inimigo" nascer num local aleatório dentro da sua "zona de *spawn*".

Instantiate

Código para colocar um objeto dentro da cena quantas vezes quiser, enquanto o aplicativo está ativo, e podendo usar condições específicas para isso ocorrer. Por exemplo, para controlar a adição dos “inimigos” no cenário de maneira automática, dentro de um tempo e de maneira controlada.

Comparetag

Código usado para verificar qual o tipo do objeto quando há uma colisão entre objetos, por exemplo, quando quero que o “jogo” saiba dizer o momento em que o “ataque” encostou no “inimigo” e realizar uma checagem de “*Tag*” para o “ataque” não acabar “derrotando” o que não deve (o cenário ou o player).

4. Desenvolvimento do jogo

O objetivo desta parte é demonstrar na prática toda a aprendizagem da ferramenta para desenvolver o jogo usado neste estudo como exemplo, e para isso é realizada uma explicação utilizando uma parte do aprendizado da Unity.

No caminho de Programador Júnior, através de vários vídeos-aula, um *dev*(desenvolvedor) explica informações básicas para criar um jogo passo a passo, e depois será tudo colocado na prática para assimilação.

O caminho todo é separado em várias unidades, nas quais cada uma irá adicionando aos poucos os elementos fundamentais que um jogo precisa para ser funcional. No fim das unidades, o conhecimento básico necessário para criar um jogo funcional é alcançado.

Será ensinado sobre controles básicos, jogabilidade básica, Sons e Efeitos, Mecânicas de jogabilidade, e Interface de Usuário.

Durante estas unidades, será dada a oportunidade de começar um projeto pessoal e adicionar o conhecimento adquirido na unidade aos poucos, criando um jogo funcional no final.

O jogo que será demonstrado é o projeto pessoal que foi desenvolvido ao longo do aprendizado desta parte.

4.1 Documentação do design do jogo

O projeto começa com o acertamento da ideia sobre o jogo. A Unity disponibiliza um documento no qual podem ser preenchidas as informações básicas sobre como o jogo será, quais os objetivos e quais as potenciais limitações.

Figura 1: Conceito do projeto, primeira parte onde é descrito o controle do jogador.

Neste caso, a primeira ideia é sobre o controle do jogador. Foi escrito que é controlado um “jogador” neste jogo de “vista lateral” onde “esquerda” e “direita” fazem o jogador “mover-se”.
Aqui o jogador pode ser qualquer coisa que imaginar, por isso a clarificação “***Player***”, o tipo do jogo pode ser entre muitos e a ação do jogador pode ser qualquer coisa, ativada por um comando ou vários comandos. Por isso que inicialmente é bom fazer um documento escrevendo o conceito básico da ideia.

Haverá uma parte para planejar cada marco do desenvolvimento e controlar quanto tempo imagina-se que será necessário para concluir.

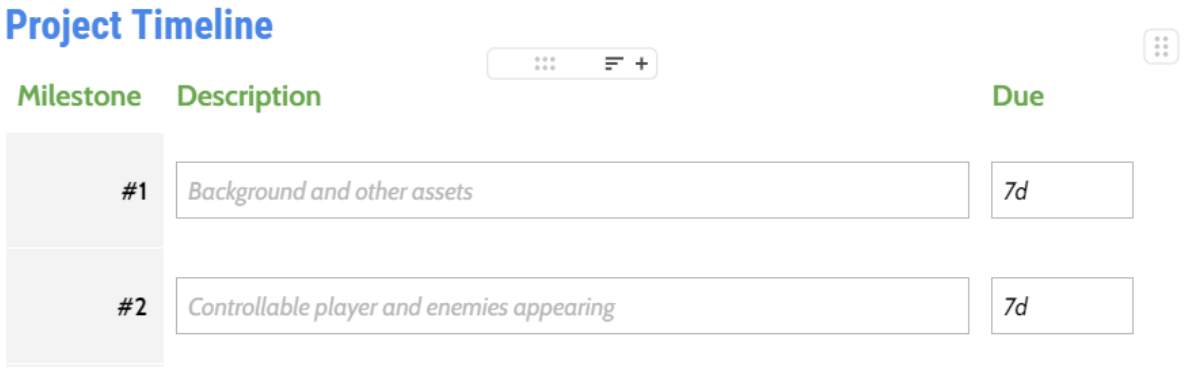


Figura 2: *Timeline* do projeto onde há as descrições das milhas e os prazos.

Aqui são divididas os marcos, as descrições e o prazo para realizá-los. Isso pode funcionar como controle e expectativa.
Por exemplo, em um tal prazo serão criados os modelos do fundo e outros assets básicos, ou, em tantas horas os controles do jogador foram desenvolvidos e os inimigos capazes de aparecer.
Esses prazos servem mais para controlar o progresso do desenvolvimento, muitos jogos não lançam na primeira data anunciada ou dentro de um prazo previsto, pois a versão atual está incompleta ou faltam alguns ajustes, e ao longo do tempo são criadas novas versões atualizadas.

No final do documento um rascunho de como o projeto é imaginado deve ser colocado para referência.

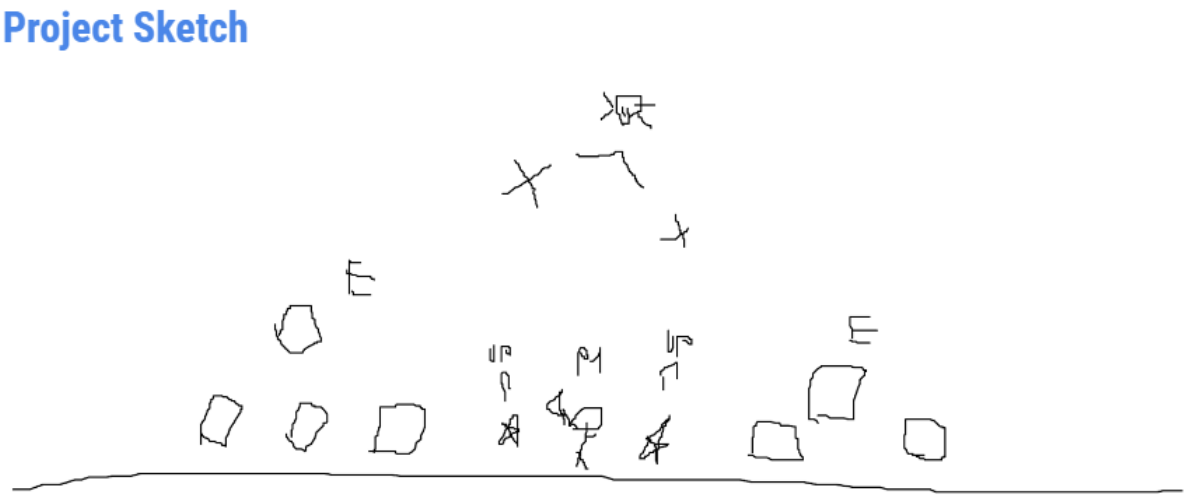


Figura 3: Esboço do projeto, uma ideia inicial de como imaginamos o produto final.

O rascunho foi feito literalmente “a mão”, muitos jogos possuem as “*concept arts*” disponíveis para ver e sempre há semelhança, ou muitas vezes é bem diferente do produto final. Um rascunho é sempre uma boa ajuda para dar uma imagem de como a ideia pode ficar.

4.2 Criando um novo projeto

Depois de ter documentado a ideia sobre o jogo, é criado um novo projeto do zero na Unity.

Este projeto começa com a criação de uma base sólida de acordo com o estilo de jogo em mente. Neste caso é criado uma “parede” de fundo, já que o jogo será na perspectiva 2D é necessário ter uma. Nota-se que mesmo não tendo uma visão 3D no produto final, ainda sim é necessário fazer tudo em 3D.

Após ter criado a “parede”, cria-se o primeiro personagem, o “jogador”, utilizando formas básicas inicialmente. É muito mais fácil e simples usar estes formatos, como uma “esfera”, para representar o jogador no início do projeto, já que serão feitas muitas alterações durante o processo. Somente depois que serão substituídas as “aparências” dos personagens.

Com o jogador criado, agora o foco é na posição da câmera. Não importa qual estilo de jogo faremos, na maioria deles o jogador é o foco principal.

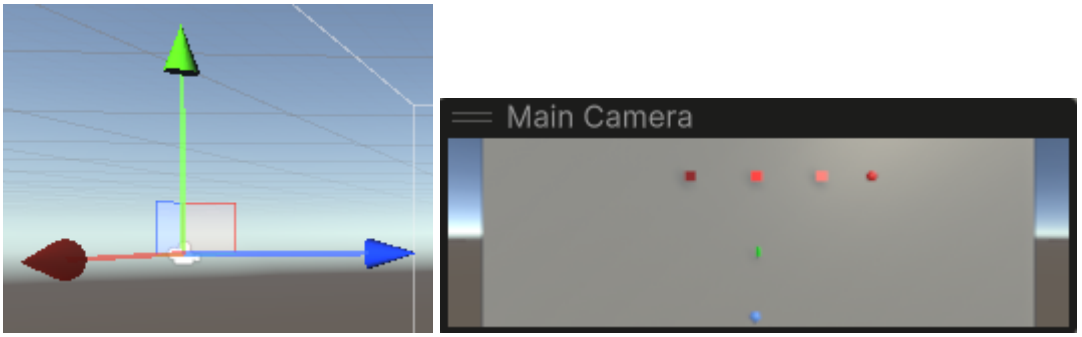


Figura 4: À esquerda, uma imagem da câmera no cenário com as opções de arrumar as dimensões dela. À direita, imagem da visão da câmera.

Após todos estes ajustes, será necessário criar os “inimigos”, “obstáculos” e “materiais” usando as formas básicas.

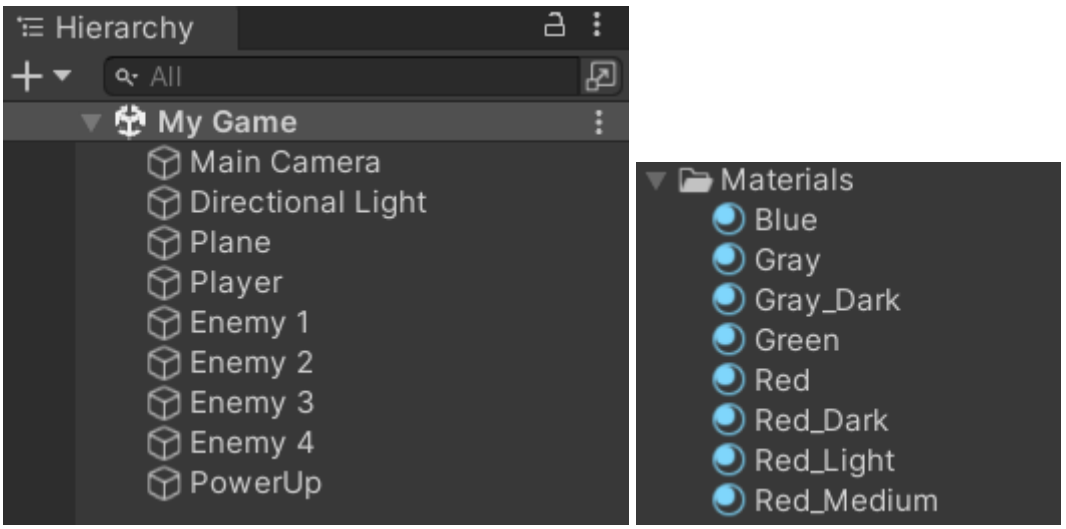


Figura 5: À esquerda, a hierarquia dos objetos. À direita, a pasta dos materiais.

Os materiais criados ajudam a identificar as diferenças entre os personagens. São usados para “pintar” as formas básicas, separando-as nos tipos de personagens que serão usados. Por exemplo, este projeto tem 3 tipos de inimigos inicialmente e todos tem um tom de cor diferente para que seja possível diferenciá-los.

Após ter criado e modificado todos os objetos, posiciona-se no cenário de maneira que faça sentido com o tipo de jogo para facilitar a visualização do projeto e um backup desta versão do projeto deve ser feito.

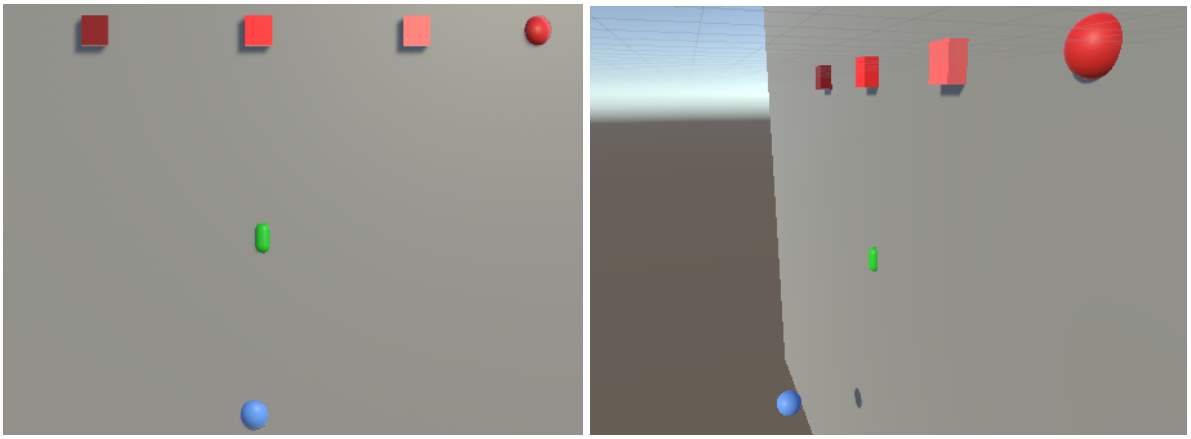


Figura 6: À esquerda, a visão inicial do cenário com os objetos em seu estado simples. À direita, um ângulo diferente para mostrar como os objetos ficam.

4.3 Aplicando controle ao jogador

Para dar controle ao jogador primeiro é dado um “componente” de “física” a ele, chamado *rigidbody*, com esse componente é possível escrever uma linha de código na qual usa-se física para mover o jogador.

Depois de ter criado a pasta de “*scripts*” e o *script* que foi nomeado de “*PlayerController*” (para saber função do *script*), o qual é usado para escrever todo o código centrado no jogador, vê-se qual tipo de programação será usada para dar o controle ao jogador e que tipo de movimentos serão dados.

Como neste caso é um jogo na perspectiva 2D pode-se focar somente na movimentação “horizontal” por hora. Como antes foi criado e aplicado o componente que aplica física ao objeto, neste caso o jogador, utiliza-se um código que “lê” o *input* das setas e “move” o boneco para a direção desejada.
Usando termos de código, o programa espera que caso o *input* “esquerda” ou “direita” seja pressionado isso faz com que uma força “empurre” o objeto na direção correspondente. Desta maneira o jogador “anda” para os lados quando a tecla correspondente é pressionada.

```
void PlayerControl()
{
    if (Input.GetKey(KeyCode.LeftArrow) || Input.GetKey(KeyCode.A))
    {
        playerRb.AddForce(Vector3.left * speed, ForceMode.Impulse);
    }
    if (Input.GetKey(KeyCode.RightArrow) || Input.GetKey(KeyCode.D))
    {
        playerRb.AddForce(Vector3.right * speed, ForceMode.Impulse);
    }
```

Figura 7: Linhas do código utilizado para dar movimento lateral ao jogador.

Para fazer o jogador “pular” é usada a mesma lógica, porém desta vez o empurrão é para “cima”. Há um problema com isso, o jogador poderá pular infinitamente e isso não é o desejado.

Para que o jogador pule somente uma vez no chão é necessário criar uma condição no código, o pulo deve somente ser possível quando o jogador estiver encostando no “chão”, com isso o problema é resolvido.

```
if (Input.GetKeyDown(KeyCode.Space) && isOnGround)
{
    playerRb.AddForce(Vector3.up * jumpForce, ForceMode.Impulse);
    isOnGround = false;
```

Figura 8: Linhas de código para fazer o jogador pular uma vez enquanto estiver no chão.

Um outro problema que surge com o movimento do personagem é ele sair do cenário 2D ou sair do foco da câmera.

Pode-se resolver isso bloqueando o acesso do jogador ao eixo X (neste caso o eixo Z é os “lados”!) e também bloquear a rotação dele para que ele não caia ao colidir ou andar. Esses bloqueios estão disponíveis dentro das opções do *RigidBody*.

Um método para barrar a passagem do jogador para fora da câmera é de criar uma parede invisível, para fazer isso basta criar um objeto e torná-lo numa parede e pô-lo onde o jogador não deve ir e por fim desmarcar a renderização da parede para torná-la invisível.

Outro método possível é utilizar um código que faça parecer que há uma parede invisível (método utilizado), é pego as coordenadas da área que irá se tornar impassável e cria-se uma condição na qual caso o jogador tente passar por ela ele será “teleportado” para onde está.

```
void PlayerRestrictions()
{
    if (transform.position.x < -xBound)
    {
        transform.position = new Vector3(-xBound, transform.position.y, transform.position.z);
    }
    if (transform.position.x > xBound)
    {
        transform.position = new Vector3(xBound, transform.position.y, transform.position.z);
    }
```

Figura 9: Código que limita a área de acesso que o jogador tem.

4.4 Jogabilidade Básica

Agora que o nosso jogador está se movendo como deve, será dado movimento ao resto dos objetos que foram criados, neste caso foram os inimigos, obstáculos e itens.

O movimento dos inimigos é baseado em um código feito para que eles sempre vão em direção ao jogador, é um cálculo feito usando as coordenadas de onde o inimigo *spawna*(nasce) e a localização atual do jogador.
Os obstáculos e itens não precisam seguir o jogador, então tudo que eles fazem é "cair" de acordo com a gravidade (pois são "pedras" caindo neste caso).

```
void Update()
{
    Vector3 lookDirection = (player.transform.position - transform.position).normalized;
    enemyRb.AddForce(lookDirection * speed);

    if(transform.position.y < boundY)
    {
        Destroy(gameObject);
    }
```

Figura 10: Código para o inimigo perseguir o jogador e ser destruído caso saia da área do jogo.

Quando finalizado todos os toques sobre os inimigos, obstáculos e itens, agora será necessário ter que adicionar um código que gerencie seus locais de spawn e como e quando vão aparecer.
Antes disso, serão transformados em "*prefabs*" (objeto "pronto" que pode ser "chamado" por código), isso serve para poder gerenciar um tipo de objeto sem ter que spawnar vários deles e para fazer versões "diferentes" do mesmo inimigo.
Para fazer isso, primeiro é criado um script nomeado "*SpawnManager*" no qual serão escritos os códigos relacionados aos *spawns*. Para fazê-los aparecer no local

planejado é preciso criar uma "zona de *spawn*" usando coordenadas determinadas, e uma função para aleatorizar o *spawn* dentro desta área. Por exemplo, o *spawn* dos inimigos é na esquerda e direita da câmera (fora da visão), no chão e no "céu", pois existem inimigos voadores.

O código é feito para que o tipo de inimigo certo spawne no local certo, e os obstáculos e itens também spawnem na área desejada.

```
void SpawnEnemyAir()
{
    float airRandomX = Random.Range(-spawnX, spawnX);
    Vector3 airSpawn = new Vector3(airRandomX, airSpawnY, -4);
    int randomAir = Random.Range(2, 4);

    Instantiate(enemies[randomAir], airSpawn, enemies[randomAir].gameObject.transform.rotation);
}
```

```
void SpawnEnemyGround() {

    Vector3[] groundSpawner = new Vector3[2];
    groundSpawner[0] = new Vector3(-spawnX, groundSpawnY, -4);
    groundSpawner[1] = new Vector3(spawnX, groundSpawnY, -4);
    Vector3 groundPosition = groundSpawner[Random.Range(0, 2)];
    int randomGround = Random.Range(0, 2);

    Instantiate(enemies[randomGround], groundPosition, enemies[randomGround].gameObject.transform.rotation);
}
```

Figuras 11 e 12: Códigos para gerenciar o *spawn* dos inimigos do solo e do ar.

Como agora o projeto possui um código executando o *spawn* de objetos o tempo todo, é preciso de outro código que faça os objetos sumirem caso não sejam mais necessários. Por hora, os obstáculos e itens podem ser deletados depois de um tempo para deixar menos pesado o aplicativo.

No caso dos obstáculos, eles vão passar pelo chão e quando fizerem isso eles serão deletados usando um *script* que lida com a colisão. Escreve-se uma linha que faça exatamente isso, quando um objeto colide com um determinado campo, neste caso quando ele "colide" com o que tem abaixo do "chão", ele é deletado. (linha de código na primeira imagem)

Quanto aos itens, basta escrever uma linha que faça eles serem deletados em alguns segundos caso o jogador não os pegue.

```
Destroy(gameObject, 1f); // destroy particle after 2 seconds
```

Figura 13: Código para destruir um objeto após 2 segundos.

Para fazer os inimigos serem “deletados” será implementado o sistema de ataque do jogador.

Neste caso o jogador é capaz de atacar para a “esquerda” ou “direita” dependendo do comando atribuído. No *script* de movimento do jogador é criado uma condição que caso fosse pressionado a tecla correspondente, ele “lançaria” o ataque. Esse ataque deve ter uma condição que conte um tempo necessário para outro ataque, sem essa condição o jogador seria capaz de atacar sem parar.

O ataque é um objeto criado e convertido em um “*prefab*”, quando o jogador “ataca” ele invoca esse “*prefab*” na forma de um ataque, neste caso uma bola de fogo para derrotar os inimigos.

```
if (Input.GetKeyDown(KeyCode.Z))
{
    Debug.Log(endTimer);
    if(endTimer <= 0)
    {
        Instantiate(attackPrefabLeft, playerRb.transform.position + new Vector3(-1.75f, 1.5f, 0),
            transform.rotation);
        endTimer = 0.7f;
```

Figura 14: Linhas de código para o ataque do jogador.

Porém nada vai acontecer sem todos os objetos terem um “componente” que detecta colisões e um *script* para acionar os códigos quando houver colisão.

Ao criar o *script* “*ColissionDetect*”, no qual escreve-se os códigos que detectam as colisões e as ações que devem ser realizadas de acordo com quem colidir com quem, atribui-se ele a todos os objetos junto com o componente.

```
private void OnTriggerEnter(Collider other)
{
    if (!other.gameObject.CompareTag("Player") && !other.gameObject.CompareTag("Ground"))
        {
        Destroy(other.gameObject);
        Destroy(gameObject);
```

Figura 15: Linhas de código para checagem de colisões do ataque do jogador.

Nestas linhas de código é verificado se o “ataque” está colidindo com um inimigo, pois suas condições são de: SE haver colisão com outro objeto e ele não ser o “Jogador” E também não for o “Chão”, ENTÃO ele destrói o outro objeto (inimigo) e se destrói (pois senão ele seria capaz de destruir a todos no caminho de uma vez e isso seria desbalanceado).

Depois de atribuir as colisões, nossos “inimigos” podem ser “derrotados” pelos ataques do jogador.

4.5 Trocando assets e dando toques finais

O jogo tem uma jogabilidade básica, e os personagens fazem o que foi idealizado que façam. Agora para deixar este projeto com uma cara mais bela e dinâmica serão usados *assets* prontos.

Caso não haja tais *assets* e muito menos o conhecimento de como criá-los, há uma alternativa para isso. Procurando por assets gratuitos disponíveis na biblioteca de *assets* da Unity, os desenvolvedores da Unity sempre encorajam a busca usando este meio para quem não possui os próprios *assets*, ou não sabem como criar, ou para quem prefere focar em outros pontos no desenvolvimento do projeto.

Para trocar os *assets*, começa-se transformando os personagens em *prefabs*. Em seguida abre-se o editor de *prefab* do objeto que será realizada a troca e arrasta-se o novo *asset* na hierarquia, é necessário escalar o tamanho, arrumar a área de colisão e a localização.

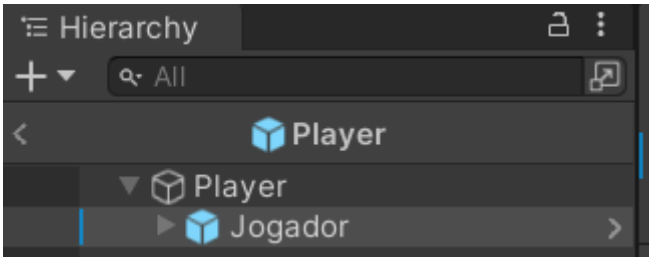
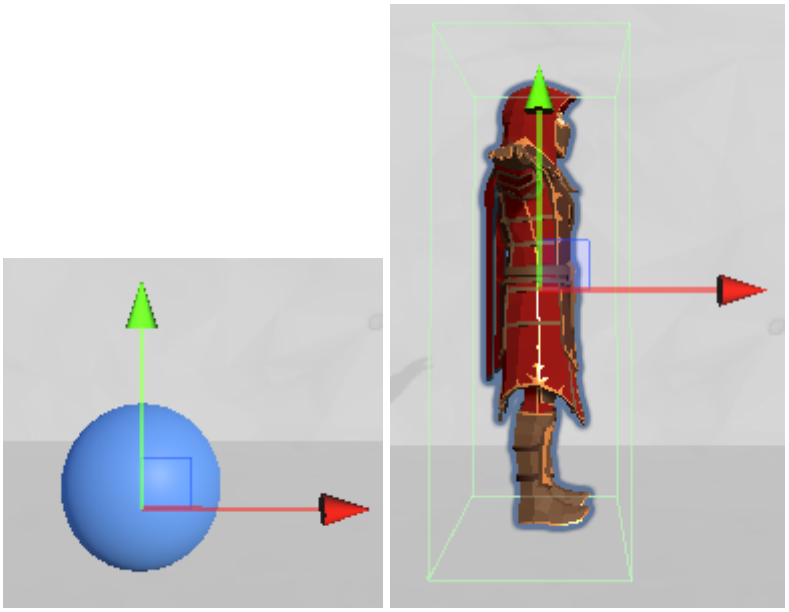


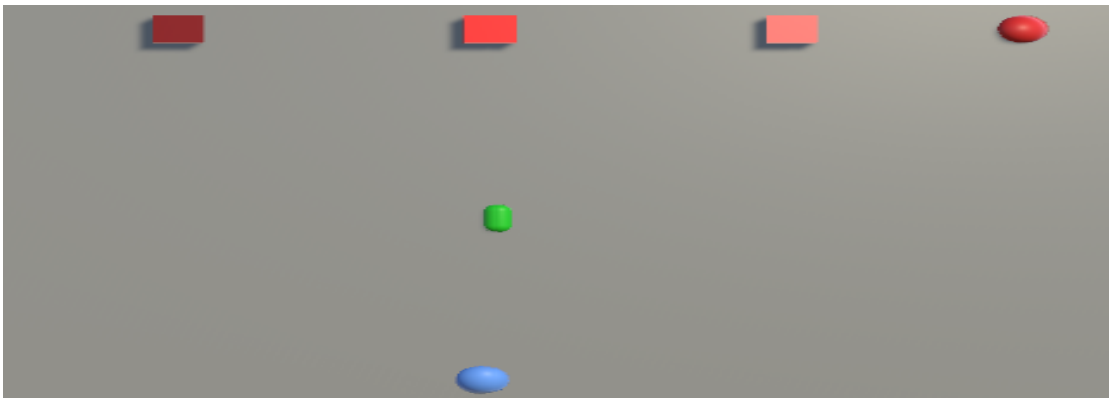
Figura 16: Resultado do jogador virar um prefab para a troca de *assets*.

Após realizar alguns testes para ter a certeza que está funcionando da maneira que deveria, agora com o novo *asset*, o antigo asset pode ser desmarcado na área de renderização para não ser visto mesclado com o novo *asset*.



Figuras 17 e 18: À esquerda, o jogador em seu estado simples. À direita, jogador após a troca de *assets*.

Ao fim desta parte, os personagens (jogador, inimigos, obstáculos e itens) terão uma nova aparência e darão a impressão de um projeto muito mais dinâmico e finalizado. O cenário (chão e fundo) também irão dar o visual cogitado do projeto.



Figuras 19 e 20: À esquerda, visão do jogo com objetos simples. À direita, visão do jogo após as trocas de *assets*.

5. Conclusão

Podemos concluir então que, para uma pessoa que busca aprender sobre o desenvolvimento de jogos eletrônicos ela possui várias opções que facilitam o processo, as *Game Engines* ajudam no desenvolvimento de jogos e até em outras áreas da indústria.

Ao adquirir mais conhecimento sobre esta tecnologia, e entender que é mais simples do que se imaginava, escolher uma *Game Engine* para usar no seu futuro projeto garante mais oportunidades e amplia suas opções.

Dentre as *Game Engines*, a Unity tem vários pontos que provam o porquê de usar uma GE. Como poder usar a tecnologia para outros meios, para melhorar o mundo e dar oportunidades aos outros.
O ensino da Unity é de qualidade e vasto, com acesso grátis e sem conhecimento prévio exigido para começar, havendo caminhos que lhe ensinam o básico até o avançado. Além de haver comunidades de todas as línguas dispostas a ajudar com seu crescimento.

Todo o processo de desenvolvimento de jogos é difícil e requer muito tempo e conhecimento em várias áreas para criar um projeto de alta qualidade. Mesmo com uma GE, muitas pessoas ainda podem ter bastante dificuldade se tentarem sozinhas, uma alternativa ideal então é focar em seus pontos fortes e deixar pessoas de confiança lidarem com seus pontos mais fracos.

Demonstrando meu projeto, transcrevo o que fui capaz de aprender com os métodos da Unity. Com meu passo-a-passo descrevendo sobre cada parte do desenvolvimento explico de maneira detalhada cada função e habilidade que adquiri.

Ao completar o conceito do projeto e a timeline de produção aprendo a como iniciar um projeto pessoal, a documentar designs, a criar uma timeline do projeto, a importância de projetar marcos e um *backlog*, além de como um rascunho da sua visão do projeto pode ajudar a imaginar o resultado desejado.

Aprendemos a como criar um novo projeto na Unity, como posicionar a câmera e perspectiva desejada e criar objetos com materiais únicos para ajudar na distinção. Com isso, sabemos o que são e como criar *primitives*, materiais e como exportar o projeto.

Com a aplicação de movimento ao jogador e as restrições necessárias aprendemos sobre programação em C# e como resolver problemas de programação que possam vir.

Para que o jogo tenha uma jogabilidade básica aprendemos a criar objetos além do jogador e atribuir seus movimentos de acordo com sua função, a transformar objetos em *prefabs* e o porquê de usá-los, criar e usar um detector de colisões, e como spawnar os objetos onde queremos, quando e em qual intervalo.

Trocamos os objetos *primitives* por novos *assets* que funcionam da mesma maneira, com isso podemos notar como o fluxo de trabalho da arte faz o projeto ter mais dinâmica e parecer mais completo, a diferença entre uma resolução baixa e alta, e que mesmo que a pessoa não seja boa em uma área ela pode focar na especialidade dela e deixar suas fraquezas com outros (por exemplo, alguém que não é bom em modelar pode pegar um *asset* grátis na loja da Unity).

Com este documento, espero ser capaz de transmitir este conhecimento e inspirar pessoas a irem atrás do sonho de criar o próprio jogo.

6. Referências Bibliográficas

Assumpção, Mariana. Descubra as 10 maiores bilheterias do cinema em 2023 (até agora). Ingresso, 2023. Disponível em: https://www.ingresso.com/noticias/lista-10-maiores-bilheterias-cinema-ano-2023?partnership=home

Barbosa Negrello, Letícia. A origem do fliperama: os primeiros jogos que fizeram história, Fala!Universidades, 2021. Disponível em: https://falauniversidades.com.br/a-origem-do-fliperama-os-primeiros-jogos-que-fizeram-historia/

Boughedda, Sam. Unity Acquires Character Design Firm Ziva Dynamics, 2022. Disponível em: https://uk.investing.com/news/stock-market-news/unity-acquires-character-design-firm-ziva-dynamics-2569443

Edwards, David. More businesses using Unity to customize designs for clients, 2022. Disponível em: https://roboticsandautomationnews.com/2022/04/27/more-businesses-using-unity-to-customize-designs-for-customers/50630/#google_vignette

Gabriel Bohnen, Júnior. Junior Tessing, Michel. Colling, Juliane. Título: JOGOS ELETRÔNICOS E SEU IMPACTO NO MUNDO: UM ESTUDO SOBRE A INTERFERÊNCIA DOS GAMES SOBRE A FORMAÇÃO DOS INDIVÍDUOS . 19 folhas. Tecnologia da Informação. FAI. Faculdades de Itapiranga. Disponível em: https://eventos.uceff.edu.br/eventosfai_dados/artigos/inova2019/1202.pdf

Junior, Wilson. As 8 Melhores Engines para Criar Jogos (Engines para Iniciantes / Games Engines Gratuitas). IlustraDev, 2022. Disponível em: https://ilustradev.com.br/melhores-engines-de-jogos-gratuitos-e-pagos/

Laurence, Felipe. Receita de games deve atingir US$ 187 bi no mundo em 2023. Valor Econômico, 2023. Disponível em: https://valor.globo.com/empresas/noticia/2023/08/08/receita-de-games-deve-atingir-us-187-bi-no-mundo-em-2023.ghtml

Luíz Duarte Pittol, Gabriel. Título: A história e contribuição dos jogos e consoles de videogame para a sociedade e a computação. 2019. 73 folhas. Ciência da Computação. UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL. RIO GRANDE DO SUL. 2019. Disponível em: https://lume.ufrgs.br/handle/10183/198493

Morales, Marie. Is Cloning the World Possible? Videogame Engine to Train AI to Effectively Handle Realistic Scenarios. TheScienceTimes, 2022. Disponível em: https://www.sciencetimes.com/articles/37505/20220505/cloning-world-possible-videogame-engine-train-ai-effectively-handle-realistic.htm

O que são as Game Engines ou Motores de Jogos? PIX STUDIOS, 2014. Disponível em: http://www.pixstudios.com.br/blog/novidades-de-computacao-grafica-e-games/o-que-sao-engine-de-games-ou-motor-de-jogo/index.html

Panya, Suhatcha. Unity Releases Cloud-Based Simulation Solution to Help Game Creators Achieve Ideal Game Balance. BusinessWire, 2020. Disponível em: https://www.businesswire.com/news/home/20200323005483/en/Unity-Releases-Cloud-Based-Simulation-Solution-Game-Creators

Peters, Jay. Unity's impressive new 'Enemies' short shows off a remarkably realistic digital human, 2022. Disponível em: https://www.theverge.com/2022/3/21/22985286/unity-enemies-short-remarkably-realistic-digital-human

Poon, Linda. How Cities Are Using Digital Twins Like a SimCity for Policymakers, 2022. Disponível em:

https://www.bloomberg.com/news/features/2022-04-05/digital-twins-mark-cities-first-foray-into-the-metaverse?sref=DJ6uFsR8

Rousseau, Jeffrey. Unity announces Unity For Humanity 2022 Grant winners. GamesIndustry, 2022. Disponível em: https://www.gamesindustry.biz/unity-announces-unity-for-humanity-2022-grant-winners

Schade, Marcos. Como iniciaram os videogames. História dos videogames, 2023. Disponível em: https://www.historiadosvideogames.com/como-iniciaram-os-video-games

Statt, Nick. Gaming could be a $300 billion industry by 2027, new report says. Protocol, 2022. Disponível em: https://www.protocol.com/newsletters/video-games-300-billion-industry

Unity. Unity Blog. Disponível em: https://blog.unity.com/industry

Unity. Unity Learn. Disponível em: https://learn.unity.com/

Unity. Unity Manual. Disponível em: Unity - Manual

Unity. Unity News. Disponível em: https://unity.com/pt/news/media

Wakka, Wagner. Mercado de games agora vale mais que indústrias de música e cinema juntas. Canaltech, 2021. Disponível em: https://canaltech.com.br/games/mercado-de-games-agora-vale-mais-que-industrias-de-musica-e-cinema-juntas-179455/