Desenvolvimento de Protótipo de um Jogo *Roguelike* Utilizando o Motor Gráfico *Unreal Engine* - Goran Týr

Matheus da S. Pires[1], **André C. da Silva**[1]

[1]Grupo de Pesquisa Mobilidade e Novas Tecnologias de Interação
Curso de Tecnologia em Análise e Desenvolvimento de Sistemas
Instituto Federal de Ciência e Tecnologia de São Paulo (IFSP)
Avenida Thereza Ana Cecon Breda, N.º 1896,
Vila São Pedro – 13183-250 – Hortolândia – SP – Brasil

matheuspires1906@gmail.com, andre.constantino@ifsp.edu.br

Abstract. *The digital gaming market has been one of the fastest-growing industries from 2020 to 2023, particularly during the pandemic years. With this in mind, the objective of this project was to develop a digital game prototype in the roguelike style, defining a story, characters, and enemies documented through the Game Design Document. The development began with the creation of the story, map, character movement, combat system, enemies, and finally the boss. Free assets were used for characters, images, backgrounds, and other elements within the game, with a focus on defining the mechanics and programming of the game.*

Resumo. *O mercado de jogos digitais é um dos que mais cresceram nos anos de 2020 a 2023, principalmente durante a época de pandemia. Considerando isso, o objetivo desse projeto foi desenvolver um protótipo digital de jogo no estilo de roguelike, definindo uma história, personagens e inimigos, registrados por meio do Game Design Document. Iniciando o desenvolvimento pela criação da história, o mapa, a movimentação, o combate, os inimigos, e por fim o chefe. Foram utilizados assets gratuitos para personagens, imagens, cenários e outros elementos dentro do jogo, focando o desenvolvimento do trabalho na definição da mecânica e sua programação.*

1. Introdução

Os jogos digitais têm como um de seus objetivos principais o entretenimento e a diversão, porém podem trazer outros benefícios como o raciocínio lógico. Segundo [Miniello 2022], durante o jogo por exemplo, é necessário pensar e ter raciocínio lógico para chegar a um determinado resultado, como: acertar a palavra em pouco tempo, evitar erros, entender a conclusão para passar de fase, entre outras coisas.

Segundo [Newzoo 2021], a indústria global de *games* movimentou US 175,8 bilhões em 2021, com base nos últimos dados consolidados. De acordo com [Pacete 2022] a indústria brasileira de *games* cresceu 169% entre os anos de 2019 e 2022.

De acordo com [Villela 2021], existem diversos tipos de jogos, também chamados de gêneros, como por exemplo ação, luta, esporte e aventura. O estilo *roguelike* é um subgênero de jogos RPG, o *Role Playing Game*. Segundo [Fonseca 2008], RPG define um estilo de jogo em que as pessoas interpretam seus personagens, criando narrativas,

histórias e um enredo. Contudo, o estilo *roguelike* é caracterizado pela geração de nível aleatório ou procedural durante a partida, cujo mapa geralmente é baseado em um cenário com uma temática de alta fantasia e morte permanente, ou seja, o avatar, ao morrer, retorna suas ações do início.

De acordo com [Saltzman 2021], *roguelike* é um jogo que apresenta tanto a geração procedural quanto a *"permadeath"* uma morte permanente, como elementos centrais do estilo de jogo, ou seja, não é apenas um modo que você tem a opção de ativar ou desativar.

O objetivo desse projeto foi desenvolver um protótipo de jogo no estilo *roguelike*, chamado Goran Týr, utilizando o motor gráfico *Unreal Engine* da Epic Games, um dos principais motores gráficos utilizado para o desenvolvimento de jogos atualmente. O jogo conta com uma história, um enredo, personagens e inimigos definidos durante o desenvolvimento. Foram utilizados *assets* gratuitos para personagens, imagens, cenários e outros elementos dentro do jogo.

O restante deste artigo está dividido em referencial teórico, apresentado na Seção 2, enquanto que trabalhos correlatos são apresentados na Seção 3. Na Seção 4 é apresentada a metodologia adotada, e o desenvolvimento do trabalho na Seção 5. A conclusão é apresentada na Seção 6 e os trabalhos futuros na Seção 7.

2. Referencial Teórico

Nesta Seção abordaremos temas relacionados a jogos digitais, explorando o seu desenvolvimento, tipos e suas características.

2.1. Como definir um Jogo?

De acordo com [Juul 2003], um jogo é um sistema formal baseado em regras, com um resultado variável e quantificável, cujo diferentes resultados são atribuídos por diferentes valores. O jogador empenha esforço a fim de influenciar o resultado, bem como sente-se vinculado e as consequências da atividade são opcionais e negociáveis.

O diálogo construído por Juul propõe seis aspectos a serem considerados na conceituação dos jogos digitais: a presença de regras, o resultado variável e quantificável, a valorização do resultado, o esforço empreendido pelo jogador, o tipo de vínculo estabelecido entre o jogador e o resultado e as consequências de suas escolhas.

Segundo [Villela 2021], o universo dos jogos tem inúmeros gêneros diferentes, cada um definido por suas mecânicas e funcionamentos distintos. MOBA (*Multiplayer Online Battle Arena*), RPG (*Role-Playing Game*), MMORPG (*Massively Multiplayer Online Role-Playing Game*) e FPS (*First-Person Shooter*) são os tipos mais populares atualmente, representados por nomes de peso, como League of Legends, Valorant e Diablo III, por exemplo.

2.2. Jogos Competitivos

Jogos Competitivos são aqueles em que há vencedores e vencidos. De acordo com [Lima 2022], o jogo competitivo pode ser caracterizado como a participação em disputas contra outros jogadores, tanto dentro quanto fora de um sistema de classificação.

Essencialmente abrange qualquer jogo no qual há a oportunidade de superar e competir diretamente com outras pessoas.

[Huizinga 1996] afirma: "quem diz competição, diz jogo". Para o autor não há razão alguma para recusar a qualquer tipo de competição o caráter do jogo. Observando tal pressuposto é exposto nesta subseção os jogos competitivos, suas características e a sua constituição em diferentes civilizações. O olhar frente a competição está associado à ideia de disputa entre indivíduos, grupos, equipes, com a finalidade a superação de limites pessoais ou coletivos. Em consequência, a vitória, a derrota e também o empate são resultados óbvios dos jogos de competição. Contudo, substancial ao jogo aceitar e respeitar qualquer resultado final.

Os jogos competitivos contam com um dos maiores rendimentos de premiações do mundo. De acordo com [Agostini 2021], os 10 jogos competitivos com os maiores rendimentos do mundo foram: Valorant, League of Legends, Rainbow Six Siege, Call of Duty: Black Ops Cold War, PUBG Mobile, Fortnite, PUBG, Arena of Valor, Counter Strike: Global Offensive e Dota 2. Somados, os 10 jogos distribuíram mais de US$ 116,7 milhões em premiações.

2.3. Desenvolvimento de Jogos

O desenvolvimento de um jogo é complexo e deve ser divido em várias partes: a criação da história e personagens, dos cenários e a ambientação e por fim as mecânicas e a jogabilidade, ou seja, o *game design* do jogo. De acordo com [Godoy and Barbosa 2010] é possível dividir o processo de desenvolvimento de uma jogo em 3 etapas: pré-produção, produção e pós-produção.

Na pré-produção é idealizado o conceito do jogo, e realizado busca por referências de mercado. A produção é a etapa onde o jogo será desenvolvido, criação do modelos dos personagens, cenários e *game design* e o primeiro protótipo do jogo. A pós-produção é iniciada após o lançamento do jogo, é etapa onde é dado suporte ao jogo, correção de problemas e atualizações.

De acordo com [Naspolini 2018], dentro do processo de desenvolvimento de jogos existem várias formas de abordagem para o desenvolvimento, como Scrum, Evolutiva, Espiral, incremental e etc. A abordagem escolhida para o desenvolvimento desse protótipo foi a incremental. A abordagem para criar esse protótipo envolve a implementação de vários ciclos curtos, nos quais funcionalidades como movimentação, combate, personagens e história são gradualmente introduzidas a cada iteração.

De acordo com [Richter 2021] *Game Design*, é a criação e planejamento dos elementos, regras e dinâmicas de um jogo. Ele é realizado pelo *Game Designer*. Dentro de uma equipe de desenvolvimento de um jogo, o *Game Designer* tem uma função importantíssima. Ele desempenhará o papel de criar as ideias do jogo, suas interações, seu enredo, suas regras e todos os elementos que deverão existir dentro do mesmo.

A prototipagem é necessária para a testagem das ideias e as mecânicas que serão incluídas no jogo, permitindo assim uma maneira mais concreta de saber se o projeto está indo pelo caminho certo ou ainda precisa de retoques. Nessa prototipagem foi utilizada a ferramenta *Unreal Engine*.

A *Unreal Engine* é um motor gráfico de criação de jogos desenvolvida pela Epic

Games. De acordo com [EpicGames 2022], ela tem sido amplamente utilizada na indústria de jogos, bem como em outras áreas, como visualização arquitetônica, design de produtos e até mesmo no cinema.

Uma *engine* é um ambiente de desenvolvimento integrado, com um conjunto pronto de ferramentas de desenvolvimento visual e componentes de software reutilizáveis, ou seja é um *framework* projetado especificamente para a construção e desenvolvimento de videogames.

De acordo [Leaf 2023], os *games assets* são os elementos essenciais que formam a base de um jogo, englobando todos os componentes visuais, auditivos e interativos que se combinam para criar a experiência imersiva que os jogadores apreciam. Cada ativo desempenha um papel distinto, contribuindo para a narrativa, a atmosfera e a jogabilidade global do jogo. Eles podem ser criados ou comprados, e existem sites onde é possível obtê-los gratuitamente, como a Mixano e a própria loja da Epic Games.

O acompanhamento desse processo de desenvolvimento do jogo é feito através do *Game Design Document* (GDD). O GDD é um documento contendo todas as informações relevantes do *design* de um jogo: temática, mecânicas, plataformas, inimigos, *levels*, entre outros. Tudo isso é importante porque o GDD servirá como um manual de montagem ou um guia na hora da criação do jogo.

De acordo com [Schell 2008], existem diversos tipos de documentos com formatos diferentes que atendem as várias necessidades, podendo ser divididos em seis grupos principais contendo suas respectivas documentações, conforme apresentado em:

1. *Design*: Resumo geral do *design* do jogo; Documento detalhado de *design*; Resumo geral da história.
2. Engenharia: Documento de *design* técnico; Resumo geral de integração de arte; Limitações do sistema;
3. Arte: Resumo geral do conceito de arte.
4. Gerência: Orçamento do jogo; Cronograma do projeto.
5. Roteiro: Bíblia da história do jogo; Script; Tutorial e manual do jogo.
6. Jogador: Acompanhamento do jogo.

De acordo com [Rollings and Morris 2004], existe uma diferença entre o GDD completo e o GDD finalizado. Um GDD completo possui os elementos necessários para produzir um protótipo "jogável" enquanto um GDD finalizado só vai existir próximo da comercialização. Para a realização desse projeto será utilizado o modelo completo, apenas com o intuito de criar um protótipo jogável.

3. Trabalhos Correlatos

A Figura 1 apresenta o jogo Hades [Hades 2020], um jogo no estilo *roguelike* em que o jogador desafia o deus dos mortos enquanto corta seu caminho para fora do submundo do mito grego.

Na Figura 2 é demonstrado o funcionamento do jogo. Nele o jogador toma o controle do protagonista chamado Zagreu, o filho do próprio deus do submundo, e sua missão é simples: fugir desse inferno. Para cumprir este objetivo, ele conta com a ajuda de seus tios e parentes do Olimpo. Goran Týr se assemelha as várias características desse jogo, porém se difere na mitologia utilizada.

Figura 1. Hades[Supergiant Games 2020]

Figura 2. *Gameplay* Hades.

A Figura 3 apresenta o jogo Dead Cells [Dead Cells 2018], definido pelos seus desenvolvedores como "um *roguelike* de ação em plataforma inspirado em Castlevania" no qual é preciso entender um combate *"souls-like"* em 2D. O jogo faz parte de um subgênero dos *games* conhecido como metroidvania. Jogos desse subgênero misturam elementos das séries Metroid e Castlevania, com a exploração não linear do mundo, variações de armas que abrem lugares secretos e uma necessidade de visitação recorrente dos ambientes. O jogo é apresentado na Figura 4.

Figura 3. Dead Cells[Motion Twin 2018]

Figura 4. *Gameplay* Dead Cells.

4. Metodologia

Durante o processo de desenvolvimento de jogos eletrônicos, existem várias etapas e processos para o seu desenvolvimento. De acordo com [Montenegro 2023], no desenvolvimento de jogos, há um ciclo geral que muitos projetos seguem, seja o responsável pelo processo de criação de grandes estúdios ou um profissional independente. Esse ciclo é composto pelas seguintes etapas: a idealização, a documentação da ideia (*game design document*), o desenvolvimento do jogo, os testes e a publicação do jogo. Devido a limitações de tempo para execução deste projeto, o objetivo é limitado a desenvolver um protótipo digital.

Algumas ferramentas foram utilizadas como a *Unreal Engine* (o motor gráfico usado para o desenvlovimento do jogo), o Mixano (um software de animação para malha dos personagens) e, por fim os, *assets* obtidos pela própria loja da Epic Games.

4.1. Idealização do jogo

A primeira atividade foi a construção da história, enredo e definição das mecânicas básicas do jogo. Em seguida foi criado os *designs* dos personagens e inimigos, definindo suas

habilidades, papéis, e suas respectivas personalidades, introduzidos dentro da narrativa. Além disso, foram selecionados *assets* disponíveis gratuitamente em portais da Internet.

Após isso, foi feita a construção da ambientação e nos cenários que compõem a história do jogo. Também foram utilizados *assets* disponíveis na Internet durante esta atividade. Por fim, foi implementada as mecânicas e a jogabilidade do jogo utilizando os processos de desenvolvimento presentes na *Unreal Engine*.

4.2. *Game Design Document* (GDD)

Para documentar todo esse processo de criação do jogo, como sua ideia, como ele funciona, qual o seu objetivo, estilo de arte, tema, mecânica, inimigos e todos as etapas de desenvolvimento, foi produzido o (GDD), composto de:

1. História;
2. Influências e Inspirações;
3. Jogabilidade e Controles;
4. Personagens.

4.3. Desenvolvimento do jogo

Neste trabalho foi utilizado o motor gráfico *Unreal Engine* da Epic Games que, de acordo com a [Ebac 2023], é um dos principais motores gráficos utilizado para o desenvolvimento de jogos atualmente. Assim, o próximo passo foi o estudo da ferramenta de desenvolvimento, o motor gráfico da Epic Games chamada *Unreal Engine*. Após o estudo foi feita a programação do protótipo considerando os cenários, personagens, a história e a jogabilidade definida nas atividades anteriores.

A *Unreal Engine* possui o *Unreal Editor*, uma poderosa ferramenta visual que permite criar e editar níveis, cenários, personagens e outros elementos do jogo.

Além disso a *Unreal* oferece uma funcionalidade chamada *Blueprints*, um sistema de nós interconectados que permite criar lógica e interações. Com uma interface gráfica intuitiva, é possível criar sistemas complexos de jogabilidade, animações, inteligência artificial e muito mais.

Durante a implementação, foi utilizado a abordagem incremental, no qual foram criados vários incrementos, onde cada incremento há adição de novas funcionalidades.

No primeiro incremento foi implementado o mapa e registrado todo o seu processo de criação. Desde a criação de um *Landscape* e um material iniciado em branco que será personalizado, a customização desse *landscape* e a criação de portais.

No segundo incremento, foi adicionado o personagem principal e sua mecânica de movimentação, animações para compor a movimentação e navegação pelo mapa.

No terceiro incremento, foi adicionado as mecânicas de combate do jogo. Como a mecânica e as animações de ataque e um sistema de notificação para contabilizar o acerto do ataque e o dano causado.

No quarto incremento, foram adicionados os inimigos. Eles contaram com um sistema de campo de visão para poder captar a presença do jogado, um sistema de perseguição, para seguir o jogador assim que for notado e por fim um sistema de ataque para causar dano a ele.

No quinto e último incremento, foi adicionado o chefe (*boss*). Nesse incremento o chefe foi adicionado e ajustado, sendo bem superior aos outros inimigos, sua vida e dano foram aumentados para intensificar a batalha. Além disso, foram adicionados dois outros inimigos junto a ele para dificultar ainda mais esse batalha.

4.4. Testes

Os teste são uma parte muito importante para o desenvolvimento e o refinamento de um jogo. De acordo com [Montenegro 2023], os testes são importantes para ajudar a equipe a encontrar erros e pontos de melhoria. Resolvê-los é essencial porque, caso fossem publicados com eles, a experiência dos usuários não seria positiva. Quanto mais testes forem feitos, mais refinado o jogo fica. Os teste para o desenvolvimento de Goran Týr foram feitos de forma manual, sendo realizados ao fim de cada incremento. Os erros e problemas provenientes deste testes foram corrigidos e novamente testados. Só se iniciou um novo incremento após o fim de todos os testes e erros.

5. Desenvolvimento do Trabalho

5.1. Definição do enredo

Goran Týr é um *roguelike*, que tem como o principal objetivo de derrotar inimigos e avançar entre o reinos, ambos baseados na mitologia Nórdica, e contar uma história ao longo desse caminho.

Em Goran Týr, o jogador tomará o controle de Goran, um jovem guerreiro de Midgard, o reino dos mortais, que teve seu vilarejo destruído e seus pais mortos por Odin. O jogador terá que enfrentar vários inimigos e chefes até conseguir chegar em Odin e concluir sua vingança.

O jogo contará com um sistema de evolução baseada em um sistema de níveis e desbloqueio de habilidades. Os jogadores ganham pontos de experiência ao derrotar certos inimigos e chefes, permitindo que eles subam de nível e aumentem seus atributos como força, agilidade ou magia e desbloqueiam novas habilidades que podem ser usados para enfrentar desafios mais difíceis. O Anexo I apresenta o GDD com a descrição do enredo e apresentação dos personagens.

5.2. Mapa

A criação de um mapa é composto por algumas etapas:

1. Criar e esculpir o *landscape*;
2. Criar e personalizar um novo material;
3. Personalizar o *landscape*;
4. Criar os portais.

As etapas são descritas detalhadamente nas subseções a seguir.

5.2.1. *Criar e esculpir o landscape*

Para criação do mapa é necessário a criação de um *landscape* (Figura 5), ou seja, uma paisagem criada com partes de terreno que são otimizadas para manter taxas de quadros

reproduzíveis em vários dispositivos diferentes. Primeiramente, no processo de criação de um *landscape*, a primeira etapa envolve a configuração do modo de exibição no formato paisagem. Isso significa que estamos planejando criar um terreno, onde a largura e a altura podem ser especificadas. A etapa seguinte envolve o processo de esculpir o terreno para criar a forma desejada. Para realizar essa tarefa, a *Unreal Engine* disponibiliza diversas ferramentas ou pincéis que permitem moldar o terreno de acordo com sua necessidade. Para esse processo foi utilizado o pincel padrão, e então esculpido o terreno com montanhas e um vale central, como evidenciado na Figura 6. Após o formato esculpido, foi criado um material para compor o *landscape*, descrito a seguir.

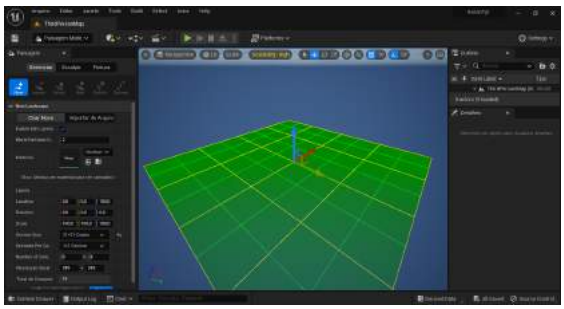

Figura 5. Landscape.

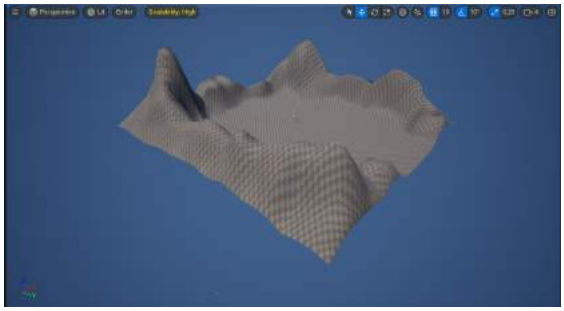

Figura 6. Landscape esculpida.

5.2.2. Criar e personalizar um novo material

Dentro da *Unreal Engine*, é possível criar e personalizar o *landscape* usando uma variedade de materiais e texturas provenientes do próprio motor gráfico. Esses materiais e texturas podem ser combinados para dar ao terreno uma aparência única e detalhada. O primeiro passo é criar esse material, onde foram selecionados as texturas e os *assets* desejados, baixados e importados para dentro do projeto. Em seguida criar um novo material personalizável.

Um novo material é iniciado em branco, sem nenhuma textura e com um *workspace*, onde é possível personalizá-lo. Dentro desse *workspace* foram selecionadas as texturas escolhidas e utilizadas para criar esse novo material. Para isso, essas texturas foram ligadas nos seus respectivos componentes e utilizado uma função de multiplicação para conseguir setar a escala do material, e por fim um nó para selecionar as coordenadas da textura. Esse processo é evidenciado na Figura 7. Depois de todo esse processo pronto, o material é criado, sendo possível concluir o *landscape* e pintá-lo, e criar uma cenário mais detalhado e ambientado com a história do jogo (Figura 8).

5.2.3. Personalizar o *landscape*

A personalização da paisagem do mapa na *Unreal Engine* é necessária para tornar o mapa mais rico e realista, sendo possível adicionar elementos como vegetação, rochas e lagos para enriquecer a aparência e a atmosfera do ambiente. Pensando nisso, foram acrescentadas textura de terra (Figura 9) e algumas folhas (Figura 10) para melhorar a customização do mapa e adicionar mais vida a ele.

Figura 7. Criação de material.

Figura 8. *Landscape* pintado.

Figura 9. Terreno sem folhagem.

Figura 10. Terreno com folhagem.

Na criação das folhas também foi utilizado *assets*. Para isso, foi selecionado o modo de *Foliage*, e escolhido as plantas que foram usadas para compor essa folhagem, e após isso foi coberto a área desejado com as plantas por meio do pincel. É possível notar a diferença entre as Figuras 9 e 10, mostrando um cenário sem a presença de folhas e outro com elas.

Depois de todas essas etapas concluídas para a produção do mapa foi necessário criar mais 5 *landscapes* e replicado esse processo nos 5 novos espaços para finalizar o mapa onde se passará a primeira parte do jogo, a região de Midgard (Figura 11).

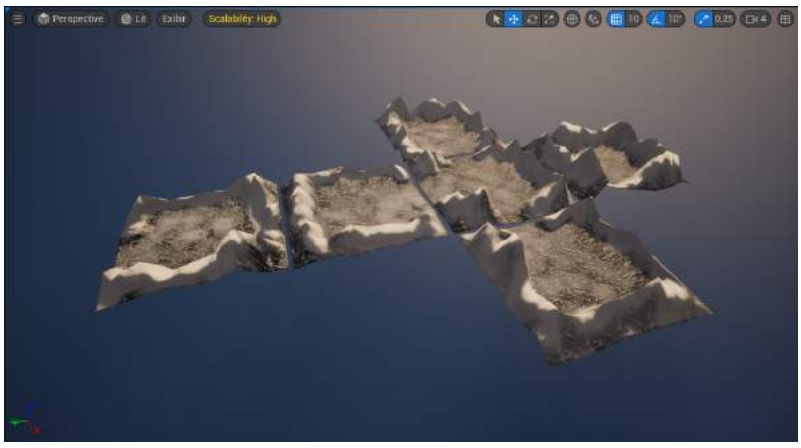

Figura 11. Mapa final de Midgard.

5.2.4. Criação de portais

Com o mapa pronto foi necessário criar uma forma de movimentar-se entre as sessões do mapa. Para solucionar este problema foi criado portais e uma mecânica de teletransporte em cada uma dessas sessões, facilitando assim essa movimentação.

Essa etapa de criação, representada pela Figura 12, foi a criação da base do portal, onde foi selecionado o formato, seu raio interno e externo, o número de partículas que o compuseram e por fim sua movimentação. O próximo passo foi a programação da mecânica de teletransporte, onde foi utilizado um *trigger* que é acionado ao passar pelo portal, chamando uma função de teletransportar, que tem como alvo o próprio jogador. Para configurar o destino do teleporte foi utilizado um ator vazio chamado *TeleporteLocation* e passando sua localização. Esse processo é demonstrado na Figura 13.

Figura 12. Base do portal.

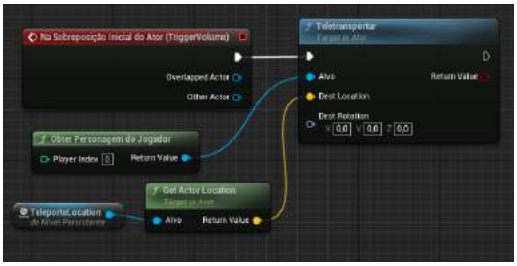

Figura 13. Teletransporte.

Ao final de tudo foram adicionados um portal em cada uma das 5 sessões do mapa, antes da sala do chefe, para melhorar a mobilidade entre salas e tornar possível o progresso do jogador no jogo. A visão final do final está exposta na Figura 14.

Figura 14. Portal.

5.3. Movimentação

Para o desenvolvimento da movimentação do personagem foi utilizado o sistema de *blueprints*. Eles são divididos em três categorias de nós:

1. Eventos: esses nós são acionados quando algo acontece no jogo, como quando um personagem colide com um objeto, quando um botão é pressionado ou quando o jogo é iniciado. Os nós de eventos definem o início de uma sequência de ações;
2. Funções e ações: esses nós realizam ações específicas, como mover um personagem, reproduzir um som, alterar a cor de um objeto, entre outros. Eles são conectados a partir de nós de eventos ou de outros nós de ação para definir a lógica do jogo;

3. Controle de fluxo: esses nós controlam a ordem em que as ações são executadas. Eles permitem ramificar a lógica com base em condições, como verificar se uma variável atende a um determinado critério antes de executar uma ação.

Sabendo disso, fica simples entender como a movimentação foi construída e o esquema é apresentado na Figura 15.

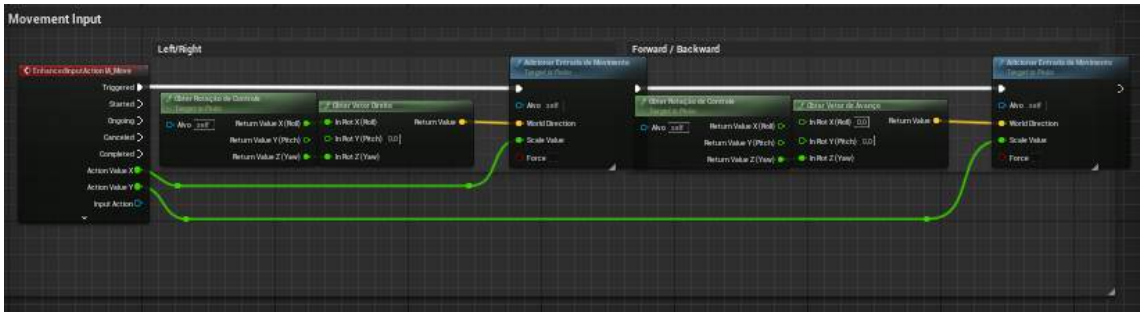

Figura 15. Esquema de movimentação do personagem.

Os nós vermelhos, ativados pelas teclas W, S, A e D, representam eventos relacionados a essas teclas e desencadeiam ações de movimentação. Os nós verdes assumem o papel de funções, responsáveis por obter informações cruciais sobre o estado do jogador. Isso inclui a localização atual no ambiente virtual, a rotação que indica a direção do olhar e o avanço do jogador. Os nós azuis, identificados como ações, desempenham papéis específicos na lógica de movimentação. Estas funções interpretam a entrada do teclado, determinam quais teclas foram pressionadas, combinam essa entrada com a posição e rotação atuais do jogador e, em seguida, calculam e aplicam a próxima posição do jogador no ambiente virtual. Após a implementação do sistema de controle de movimentação, foi necessário criar um esquema para animar os diferentes estados de movimento do personagem. Esse esquema é conhecido como *blendspace* e envolve a combinação de três animações distintas: uma para o personagem parado, outra para o personagem andando e uma terceira para o personagem correndo. A Figura 16 mostra como funciona um *blendspace*.

Figura 16. *Blendspace* do personagem parado.

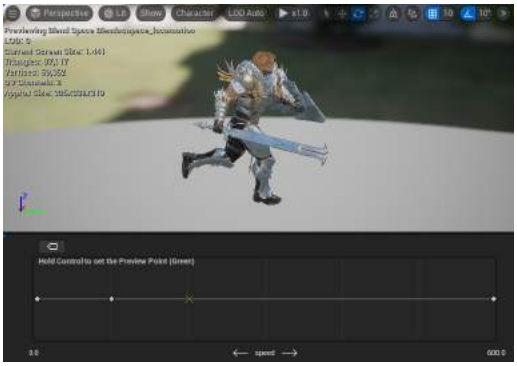

Figura 17. *Blendspace* do personagem em movimento.

Para incorporar as animações à malha do personagem, foi preciso criar um *blueprint* de animação. Nele, o *blendspace* previamente criado é selecionado e integrado.

Sendo necessário criar uma variável chamada *ground speed* para setar a velocidade da animação, para que ela seja compatível com a velocidade da malha do personagem. O *blueprint* de animação atua como um conjunto de instruções que orientam como as animações devem ser aplicadas à malha do personagem. A Figura 18 demonstra a lógica utilizada na animação do personagem utilizando o *blendspace* criado para ele.

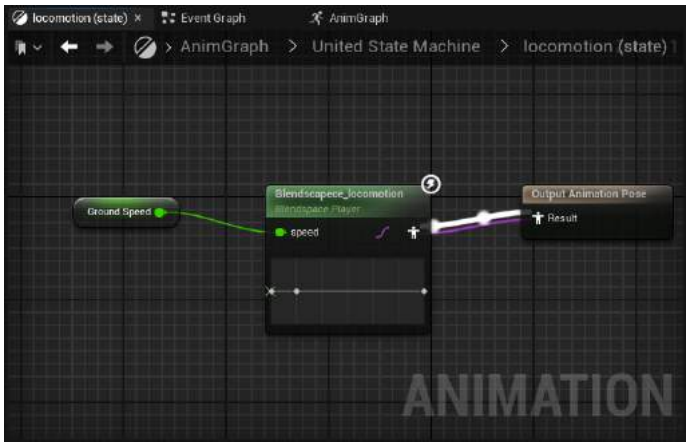

Figura 18. *Blueprint* de animação do personagem.

5.4. Combate

5.4.1. Mecânica de Ataque

A primeira etapa para criar o combate foi selecionar o animação do ataque principal do personagem e criar 2 marcadores para verificar se houve contato com algum inimigo, chamados *sockets*. Para o protagonista, especificamente, esses *sockets* são criados na mão que segura a espada. Eles foram nomeados como *SwordAttackSocketStart* e *SwordAttackSocketEnd*, representando o início e o fim da espada durante o ataque. Esses *sockets* desempenham o papel dos marcadores, permitindo que o sistema identifique e avalie o contato com inimigos durante a animação do ataque principal do personagem. A Figura 19 mostra a criação do *socket SwordAttackSocketEnd* e seu posicionamento dentro da animação de ataque do personagem.

Figura 19. *Socket* colocado na ponta da espada.

Na implementação simplificada, a detecção e comunicação de um ataque bem-sucedido são gerenciadas por um *blueprint* denominada *NotifyState*. Esse *blueprint* recebe dois *sockets* como argumentos, representando visualmente uma linha imaginária que

se forma durante toda a animação do ataque. Durante esse processo, os dois *sockets* permanecem interligados, criando uma espécie de linha virtual.

A lógica por trás disso é que, se essa linha virtual atravessar a área ocupada por um inimigo durante a execução da animação, o *blueprint NotifyState* notificará sobre o sucesso do ataque. Em resumo, o *blueprint* utiliza essa linha imaginária para identificar e informar quando o ataque atinge seu alvo durante a animação. A lógica por trás do sistema de notificação está representado na Figura 20.

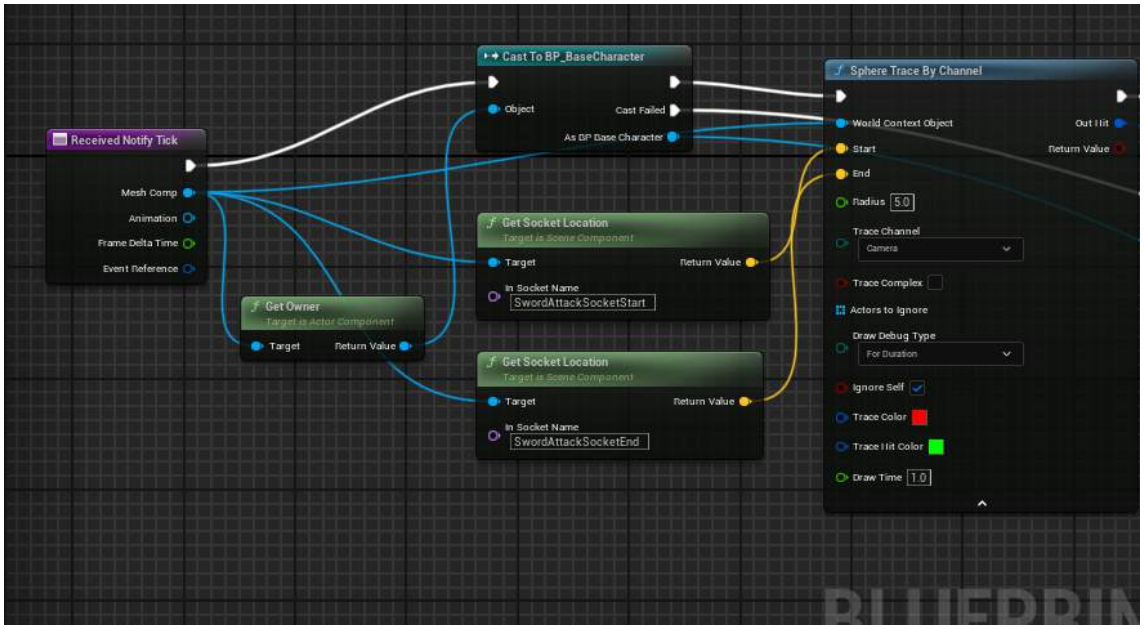

Figura 20. *Blueprint* de notifição do ataque.

Depois de concluir todo esse processo, tornou-se essencial implementar uma maneira de aplicar dano assim que o protagonista acertar o inimigo, e vice-versa. Vale ressaltar que tanto os *blueprints* do protagonista quanto dos inimigos são derivados de uma classe pai chamada *BPBaseCharacter*. Isso significa que todas as funções e variáveis criadas nesse *blueprints* principal são herdadas pelos seus descendentes.

Para esse processo, foi criado uma variável chamada *health* para monitorar a vida dos personagens. A partir desse ponto, foi estabelecido um evento denominado *AnyDamage*, que é ativado imediatamente ao detectar um contato durante um ataque. Dessa forma, o dano do ataque é subtraído diretamente da vida do personagem por meio dessa variável *health*. Esse processo continua até que a vida do personagem atinja zero, momento em que ele é considerado derrotado. Esse *blueprint* é exposto na Figura 21.

5.4.2. Esquiva

Uma mecânica essencial para o sistema de combate envolve a capacidade de esquivar-se dos ataques inimigos. Para implementar essa funcionalidade, foi introduzida a esquiva, que utiliza tanto uma animação, Figura 22, quanto um evento para realizar essa ação.

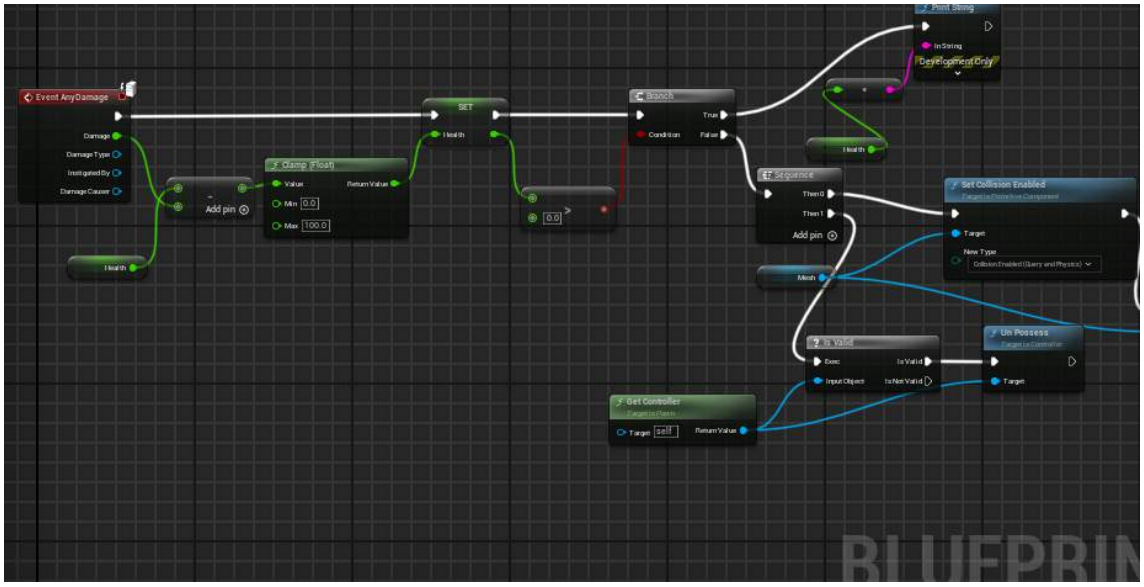

Figura 21. *Blueprint* da aplicação do dano.

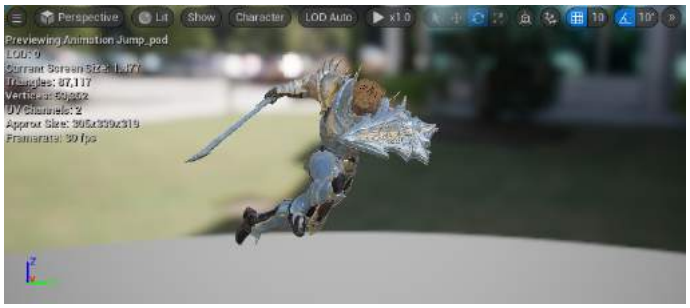

Figura 22. Esquiva.

A abordagem para desenvolver a mecânica de esquiva se concentra em duas variáveis-chave. A primeira, chamada *canDash?*, impede que o jogador execute esquivas consecutivas, garantindo um limite para essa ação. A segunda variável, denominada *isDashing*, serve para verificar se o jogador está atualmente executando a esquiva.

Na prática, para realizar uma esquiva, o evento correspondente é ativado e passa por uma verificação para determinar se o jogador tem permissão para executar essa ação naquele momento. Se a verificação for positiva, a esquiva é executada, e um temporizador é acionado para marcar o período em que o jogador não poderá realizar outra esquiva. Caso a verificação seja negativa, nenhuma ação é tomada. Todo esse processo é evidenciado na Figura 23.

5.5. Inimigos

Conforme mencionado anteriormente, tanto o protagonista quanto os inimigos derivam de um *blueprint* pai denominado *BPBaseCharacter*. Isso implica que todas as funcionalidades, como a gestão da vida e o processo de receber e aplicar danos, já estão incorporadas nos blueprints de cada personagem. Dessa forma, evita-se a necessidade de retrabalho, uma vez que tais características foram herdadas do *blueprint* principal.

O que difere os inimigos do protagonista é a inclusão de um campo de visão

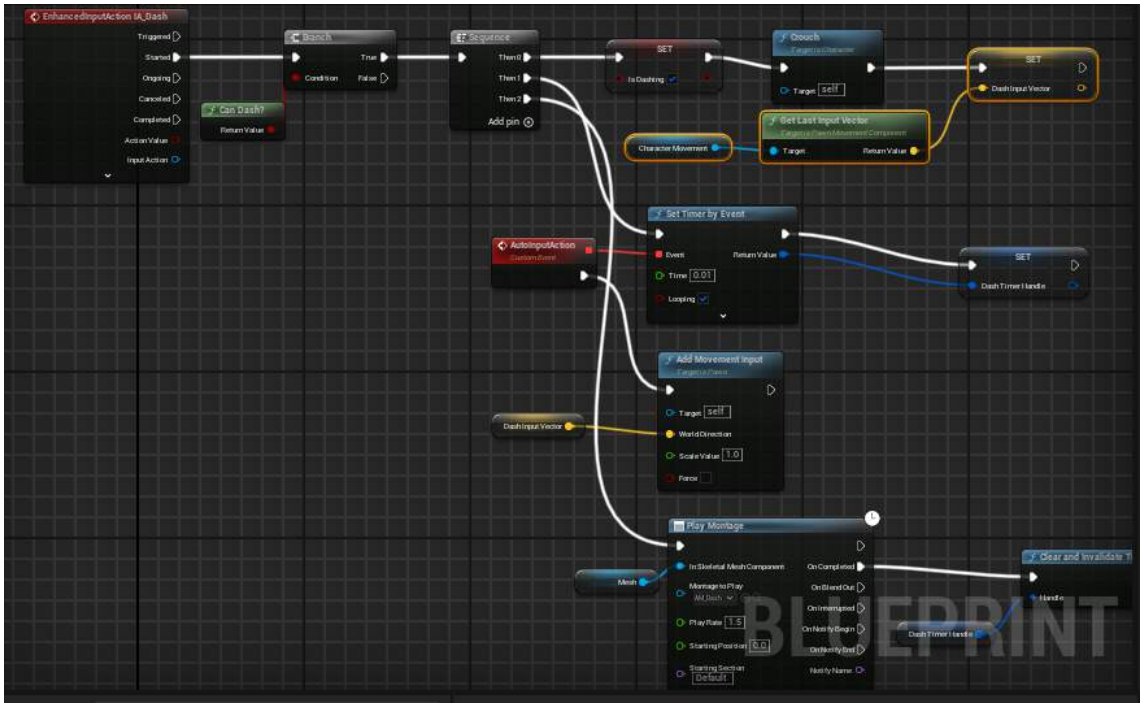

Figura 23. *Blueprint* da lógica da esquiva.

em sua malha de personagem, juntamente com a mecânica de perseguição ao jogador assim que este adentra seu campo de visão. Uma vez alcançado, o inimigo passa a atacar o jogador. O campo de visão é denominada *PawnSeeing*, que permite que o inimigo perceba a presença do jogador. Na Figura 24 é possível ver o campo de visão do inimigo, representado pelo cone verde.

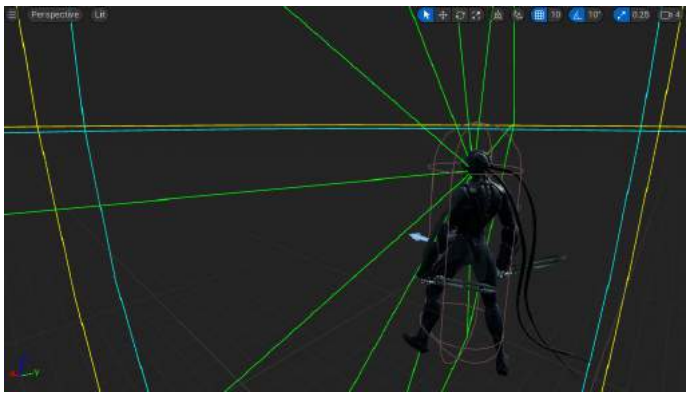

Figura 24. Campo de visão do inimigo.

Na implementação da perseguição, foi criado a variável *isSeeing?* para verificar se o jogador entrou no campo de visão do inimigo. Assim que o inimigo percebe a presença do jogador, essa variável aciona a função *FollowPlayer*, encarregada de perseguir jogador. O próximo passo, foi criar outra variável *canAttack?* para controlar a possibilidade do inimigo atacar várias vezes em sequência, como mostrado na Figura 25. Com essa etapa finalizada, ao alcançar o jogador, o inimigo executa o ataque, seguido por um

curto intervalo de tempo antes de poder atacar novamente.

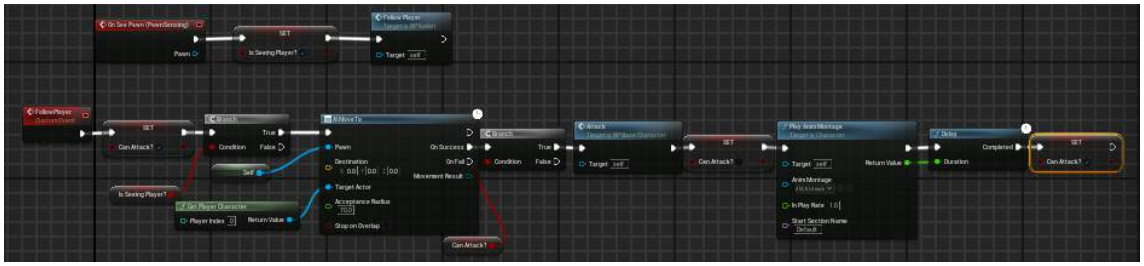

Figura 25. Perseguição e ataque.

Assim como descrito anteriormente no processo de animação do protagonista, a animação do movimento dos inimigos exigiu a criação de um *blendspace* e uma *blueprint* de animação para cada um, seguida das devidas configurações para cada um deles. No caso do ataque, o procedimento foi semelhante: escolher a animação adequada, criar os *sockets* para notificar o dano e incorporá-los ao personagem. É importante ressaltar que foram criado 3 inimigos diferentes utilizando 3 *assets* obtidos na loja da Epic Games e expostos no Anexo I, porém só foi mostrado o desenvolvimento de um, pois todos funcionam da mesma maneira. Depois de concluir o desenvolvimento do inimigo, é crucial permitir sua movimentação inteligente no ambiente do jogo. Isso é alcançado através da criação de uma área designada para calcular as rotas de movimentação possíveis e restringir a área de atuação do inimigo, conhecida como *NavMeshBoundsVolume*, atuando como uma espécie de "mapa"para os inimigos, delineando os limites do espaço onde eles podem se mover. A área verde representada na Figura 26 mostra todo o espaço de movimentação dos inimigos.

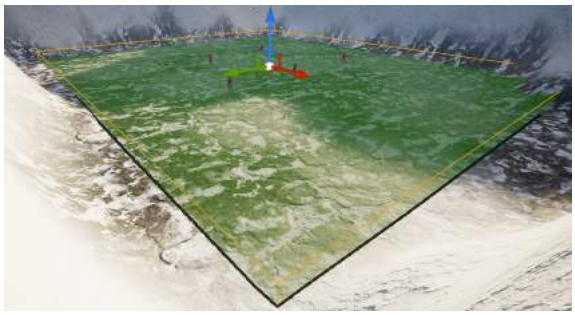

Figura 26. Campo de movimentação.

5.6. *Boss*

A criação do chefe (*boss*) seguiu o mesmo padrão utilizado para os inimigos, incorporando elementos como *blendspace* e uma *blueprint* de animação, assim como a configuração de combate permitindo que o chefe receba e cause danos, além de incluir a funcionalidade de perseguição, similar aos inimigos comuns.

No entanto, sendo o desafio final da fase, foram implementadas algumas distinções. A vida do chefe foi consideravelmente aumentada para 10.000 pontos, enquanto seu dano base foi ajustado para 500, representados pelas Figuras 27 e 28.

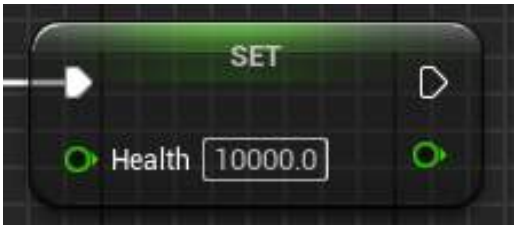

Figura 27. Vida do *boss*.

Figura 28. Dano do *boss*.

Além disso, para intensificar o confronto, foram adicionados mais dois inimigos que acompanham o chefe (Figura 29).

Figura 29. Luta final.

5.7. Renascer

A última mecânica implementada nesse protótipo foi a de renascer. Quando o jogador for derrotado por um inimigo no jogo, ele renasce e retorna ao início do jogo. A partir desse ponto, enfrentará novamente os desafios dos inimigos.

Para implementar essa funcionalidade, dentro do modo de jogo, foi criado o evento *Respawn Player*, esse evento envolve a passagem da *blueprint* do ator que será ressuscitado. Em seguida, aciona-se a função *Possess*, responsável por devolver o controle do personagem ao jogador após sua morte. A Figura 30 mostra o evento de *Respawn Player* criado no modo de jogo.

Figura 30. Evento de renascimento.

O próximo passo foi criar um ator simples chamado *BPRespawn*. Ele será utilizado para marcar a posição de renascimento do jogador. Na sequência, dentro da *blueprint* do protagonista, foi configurado o evento *destroyed*, que é acionado quando o corpo do protagonista é eliminado após sua morte. Esse evento, por sua vez, ativa a função *Respawn Player*, passando a posição do ator *BPRespawn* como parâmetro. Esse processo está representado na Figura 31. Ao final o jogador é revivido e reaparece no início do jogo.

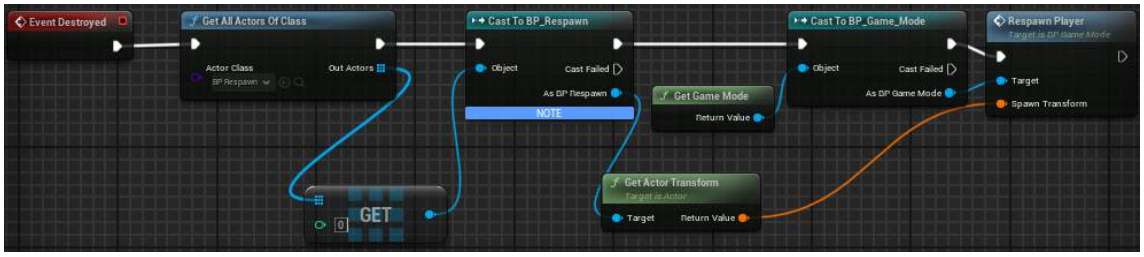

Figura 31. Lógica do renascimento.

6. Conclusão

A proposta deste projeto foi criar um protótipo de jogo no estilo *roguelike*, com ênfase na mitologia nórdica, utilizando a *Unreal Engine*, um motor gráfico da Epic Games. A etapa inicial envolveu a avaliação da viabilidade desse jogo, incluindo a análise de outros dois jogos que compartilham o mesmo estilo. Após essa fase, foi desenvolvida uma narrativa e um enredo, ambos incorporados à construção do GDD. E por fim o desenvolvido do protótipo utilizando o motor gráfico *Unreal Engine*.

A principal proposta de Goran Týr é destacar a construção de um jogo no estilo *roguelike*, onde os esforços empreendidos pelo jogador serão direcionados a estratégia, a tomada de decisões precisas, e em seu tempo de resposta, guiando assim o rumo de suas batalhas e de toda sua jornada durante a trama.

A execução deste projeto demandou um longo período dedicado ao estudo da *Unreal Engine*, o motor gráfico utilizado para conduzir o desenvolvimento do protótipo. Além disso, foi essencial contar com o suporte de um *software* externo, como o Mixano, para facilitar e aprimorar a fase de animação dos personagens. Este processo combinado permitiu uma abordagem mais abrangente e eficaz na criação do jogo, integrando tanto os recursos avançados da *Unreal Engine* quanto as capacidades do Mixano para animações.

O desenvolvimento deste projeto foi moldado pelos aprendizados adquiridos nas disciplinas de Linguagem de Programação, Algoritmos e Lógica de Programação e Estrutura de Dados. Esses conhecimentos desempenharam um papel crucial na elaboração dos *blueprints*, fornecendo a base essencial para a criação das estruturas lógicas do jogo. A aplicação de metodologias de pesquisa científica e tecnológica, por sua vez, foi um alicerce sólido na construção do artigo, permitindo uma abordagem fundamentada e estruturada. Por fim, a adoção das Metodologias Ágeis guiou o progresso e gerenciamento do projeto.

7. Trabalhos Futuros

Com a estrutura do protótipo estabelecida, torna-se viável aprimorar tanto a movimentação quanto o sistema de combate. É possível introduzir uma mecânica defensiva para os personagens, proporcionando aos jogadores a capacidade de se protegerem. Além disso, a inclusão de um sistema de cura permitirá que os jogadores restaurem sua saúde. A implementação de uma barra de vida para os inimigos, aumentando a compreensão do jogador sobre a situação em combate também é uma opção viável.

Além disso, o sistema de aprimoramento do personagem, permitindo a expansão do leque de habilidades e de seus ataques, visando intensificar os desafios enfrentados pelos jogadores. Esta inclusão não apenas diversifica as opções táticas, mas também eleva o nível de complexidade do jogo. Finalmente, desenvolver um sistema que não apenas randomize a disposição das salas, mas também introduza aleatoriedade na geração de inimigos em níveis mais avançados.

Referências

Agostini, M. (2021). Veja os 10 jogos competitivos com os maiores rendimentos do mundo em 2021. Disponivel em: https://www.techtudo.com.br/listas/2021/10/veja-os-10-jogos-competitivos-com-os-maiores-do-mundo-em-2021-esports.ghtml. [Online; Acessado em: 20/02/2023].

Dead Cells (2018). Motion Twin. Disponivel em: https://dead-cells.com/ [Online; Acessado em: 20/03/2023].

Ebac (2023). Unity x unreal: qual é o motor de jogos mais adequado para um projeto? Disponivel em: https://ebaconline.com.br/blog/unity-unreal-qual-e-o-motor-de-jogos-mais-adequado-para-um-projeto. [Online; Acessado em: 16/06/2023].

EpicGames (2022). Tempo real para todos. Disponivel em: https://www.unrealengine.com/pt-BR/solutions/more-uses. [Online; Acessado em: 16/03/2023].

Fonseca, W. (2008). O que é RPG? Disponivel em: https://www.tecmundo.com.br/jogos/243-o-que-e-rpg-.htm. [Online; Acessado em: 16/06/2023].

Godoy, A. and Barbosa, E. (2010). Game-scrum:an approach to agile game development. Disponivel em: http://sbgames.org/papers/sbgames10/computing/short/Computing_short19.pdf. [Online; Acessado em: 16/04/2023].

Hades (2020). Supergiant Games. Disponivel em: https://www.supergiantgames.com/games/hades/ [Online; Acessado em: 20/03/2023].

Huizinga (1996). *Homo Ludens: o jogo como elemento de cultura. São Paulo: Perspectiva.* 4th edition.

Juul, J. (2003). The game, the player, the world: Looking for a heart of gameness. Disponivel em: http://www.jesperjuul.net/text/gameplayerworld/. [Online; Acessado em: 20/02/2023].

Leaf, M. (2023). O que são assets na indústria de jogos? Disponivel em: https://mainleaf.com/pt/o-que-sao-assets/#:~:text=Os%

20assets%20s%C3%A3o%20os%20blocos%20de%20constru%C3%A7%C3%A3o%20fundamentais,a%20narrativa%2C%20a%20atmosfera%20e%20a%20jogabilidade%20final. [Online; Acessado em: 17/10/2023].

Lima, A. (2022). O que são jogos competitivos. Disponivel em: https://segredosdomundo.r7.com/jogos-competitivos/. [Online; Acessado em: 17/10/2023].

Miniello, B. (2022). Benefícios dos jogos eletrônicos para a memória e o cérebro. Disponivel em: https://drbrunominiello.com.br/beneficios-dos-jogos-eletronicos-para-a-memoria/. [Online; Acessado em: 24/03/2023].

Montenegro, B. (2023). Como criar um jogo?. Disponivel em: https://ebaconline.com.br/blog/como-criar-um-jogo. [Online; Acessado em: 20/03/2023].

Naspolini, F. (2018). Metodologias para desenvolvimento de jogos. Disponivel em: https://www.fabricadejogos.net/posts/metodologias-para-desenvolvimento-de-jogos/. [Online; Acessado em: 17/10/2023].

Newzoo (2021). The global game market will grow from us175.8 billion in 2021 to more than us200 billion in 2024. Disponivel em: https://newzoo.com/resources/trend-reports/newzoo-global-games-market-report-2021-free-version. [Online; Acessado em: 24/11/2022].

Pacete, L. (2022). Em quatro anos, indústria brasileira de games cresceu 169%. Disponivel em: https://forbes.com.br/forbes-tech/2022/07/em-quatro-anos-industria-brasileira-de-games-cresceu-169/. [Online; Acessado em: 16/06/2023].

Richter, P. (2021). O que é game design e o que faz um game designer. Disponivel em: https://gamedesign.com.br/o-que-e-game-design-e-o-que-faz-um-game-designer/. [Online; Acessado em: 20/03/2023].

Rollings, A. and Morris, D. (2004). *Game Architecture and Design - A new edition.* New Riders Publications.

Saltzman, M. (2021). The best roguelike games. Disponivel em: https://www.ign.com/articles/top-10-roguelikes. [Online; Acessado em: 16/05/2023].

Schell, J. (2008). *The Art of Game Design A Book of Lenses*. Morgan Kaufman Publishers,.

Villela, M. (2021). Moba, rpg, mmorpg, fps e mais: entenda significado dos gêneros de games. Disponivel em: https://www.techtudo.com.br/noticias/2021/03/moba-rpg-mmorpg-fps-e-mais-entenda-significado-dos-generos-de-games.ghtml. [Online; Acessado em: 24/02/2023].

8. Anexo I - GDD

8.1. História

Goran era um jovem guerreiro nascido em um pequeno vilarejo em Midgard, uma terra repleta de deuses e criaturas mitológicas. Quando Goran era apenas uma criança, seu vilarejo foi destruído por uma invasão liderada por Odin, o todo-poderoso deus nórdico. Seus pais foram mortos diante de seus olhos e deixou Goran para morrer.

Goran cresceu com um único objetivo: vingar-se de Odin e derrubar Asgard. Ele sabia que isso seria uma tarefa difícil, mas estava disposto a enfrentar qualquer desafio para atingir seu objetivo.

Para enfrentar Odin, Goran precisaria derrotar Heimdall, guardião da Bifrost, a ponte utilizada para viajar entre os reinos, e assim chegar até Asgard. Ele começou sua busca em Midgard, um reino de fogo onde Baldur, o deus da luz, realizava algumas tarefas para Odin.

Goran viu uma oportunidade de mandar um recado a Odin, derrotando alguém de sua família. Durante o combate Goran foi ferido mortalmente, porém, Goran tinha um poder oculto que somente Odin sabia, ele era o filho de Hella, a deusa da morte e traria a ruína a Asgard. Com esse poder desperto, Goran foi capaz de derrotar Baldur com extrema facilidade.

Em seguida, Goran viajou por Álfheim, Jotunheim e Niflheim, até encontrar Heimdall em Svartalfheim, o reino dos elfos negros, onde o enfrentou. Heimdall possuía uma espada que podia cortar qualquer coisa, mas Goran mais uma vez liberou seu poder e matou Heimdall. Quanto mais usava seu poder mais forte e corrompido Goran ficava.

A caminho de Asgard, Goran foi parado por Thor em Helheim, o reino dos mortos. Odin mandou Thor para matar Goran antes que chegasse a Asgard. Porém ele já estava forte o suficiente para enfrentar o deus do trovão. E saiu vitorioso dessa batalha.

Finalmente em Asgard. Goran enfrentou Odin em uma batalha épica que decidiria seu destino. Goran lutou com tudo o que tinha e, no final, emergiu vitorioso. Ele finalmente havia alcançado sua vingança contra Odin.

8.2. Influências e Inspirações

O estilo de jogo *roguelike* é um subgênero de jogos RPG, com geração de niveis aleatoriamente e com morte permanente do jogador. Por conta disso, o jogador irá morrer várias vezes até completar o jogo pela primeira vez. Contudo isso abre uma gama de possibilidades e formas distintas para o jogador tentar finalizar o jogo.

Títulos como Hades, The Binding of Isaac e Into the Breach seguem essa mesma mecânica de morte permanente e o recomeço do jogo até o jogador concluir seu objetivo de chegar ao final do jogo.

8.3. Jogabilidade e Controles

- A movimentação da camera é feita pelo mouse.
- A movimentação do personagem é realizada através das teclas W,A,S,D.
- O ataque do personagem é feito pelo botão esquerdo do mouse.
- A esquiva é realizada através da tecla shift.

8.4. Personagens

Figura 32. Goran[Protagonista].

Goran, um exímio espadachim, domina com maestria a arte do escudo, combinando habilidades formidáveis em ambas as disciplinas. Órfão desde pequeno, sua infância foi marcada por uma única e ardente aspiração: vingar a morte de seus pais.

Figura 33. Kallari[Inimigo em Midgard].

As Kallari são guerreiras robóticas concebidas por meio de magia obscura, constituindo uma parte integral do vasto exército de Odin. Sua criação resultou de uma fusão entre ciência e feitiçaria, dotando-as de habilidades excepcionais e uma lealdade inquebrantável ao deus supremo.

Figura 34. Khaimera[Inimigo em Midgard].

Os Khaimeras constituem uma raça guerreira ancestral de Midgard, contudo, restam apenas alguns membros de sua espécie. Ao longo dos tempos, esta linhagem robusta e combativa enfrentou inúmeros desafios, resultando em uma população significativamente reduzida.

Figura 35. Terra[Inimigo em Midgard].

As guerreiras Terra integram a distinta Guarda Real de Odin, submetendo-se a um treinamento desde a infância com o propósito singular de proteger o palácio real.

Figura 36. Baldur[Boss em Midgard].

Descendente de Odin e Frigg, a deusa que preside sobre o casamento e a maternidade, Baldur emergiu em Midgard como o primeiro e imponente desafio na jornada de Goran.

Protótipo finalizado, mostrando como ficou a jogabilidade, os cenários e inimigos dentro do jogo. Figura 37.

Figura 37. Goran Týr.

Documento Digitalizado Público

TCC - Anexo I - versão final do artigo

Assunto: TCC - Anexo I - versão final do artigo
Assinado por: Andre Constantino
Tipo do Documento: Relatório
Situação: Finalizado
Nível de Acesso: Público
Tipo do Conferência: Documento Digital

Documento assinado eletronicamente por:

- **Andre Constantino da Silva**, **PROFESSOR ENS BASICO TECN TECNOLOGICO**, em 01/02/2024 16:56:36.

Este documento foi armazenado no SUAP em 01/02/2024. Para comprovar sua integridade, faça a leitura do QRCode ao lado ou acesse https://suap.ifsp.edu.br/verificar-documento-externo/ e forneça os dados abaixo:

Código Verificador: 1563379
Código de Autenticação: 033cd0b390