Stephanie Miho Urashima

Syncing Music and Gameplay

São Paulo, SP

2023

Stephanie Miho Urashima

Syncing Music and Gameplay

Trabalho de conclusão de curso apresentado ao Departamento de Engenharia de Computação e Sistemas Digitais da Escola Politécnica da Universidade de São Paulo para obtenção do Título de Engenheiro.

Universidade de São Paulo – USP

Escola Politécnica

Departamento de Engenharia de Computação e Sistemas Digitais (PCS)

Orientador: Prof. Dr. Ricardo Nakamura

São Paulo, SP

2023

Catalogação-na-publicação

Urashima, Stephanie Miho
Syncing Music and Gameplay / S. M. Urashima -- São Paulo, 2023.
65 p.

Trabalho de Formatura - Escola Politécnica da Universidade de São Paulo. Departamento de Engenharia de Computação e Sistemas Digitais.

1.JOGOS DE COMPUTADOR 2.ENGENHARIA DE SOFTWARE I.Universidade de São Paulo. Escola Politécnica. Departamento de Engenharia de Computação e Sistemas Digitais II.t.

Agradecimentos

Agradeço ao Prof. Dr. Ricardo Nakamura pelo acompanhamento como orientador do projeto durante o desenvolvimento do projeto.

Agradeço a minha famılia, amigos e amigas pelo apoio durante a graduação. Especialmente a Lígia de Castro por auxiliar na composição da trilha sonora do protótipo realizado

Resumo

Este trabalho tem como objetivo explorar a sincronização de música e jogabilidade em jogos, investigando técnicas utilizadas em game engines e middlewares de áudio. A motivação por trás deste estudo é a necessidade de compreender a sincronização de música e jogabilidade em jogos, a fim de oferecer uma experiência imersiva aos jogadores. Para alcançar esse objetivo, foi realizada uma pesquisa geral sobre o estado da arte em sincronização em jogos, com foco em jogos de ritmo por serem mais exigentes nesse aspecto. A metodologia de trabalho envolveu a análise de técnicas utilizadas em game engines e bibliotecas de áudio, bem como a definição de requisitos funcionais e não funcionais. Como resultado, foi desenvolvido uma prova de conceito de jogo com sincronização de música e jogabilidade em tempo real, utilizando uma arquitetura especificada. Este estudo contribui para a compilação e verificação formal de informações sobre sincronização de música e jogabilidade em jogos, visando melhorar a compreensão e aplicação dessa técnica na indústria de jogos.

Palavras-chave: Sincronização, música, jogabilidade, game engines, Fmod.

Abstract

This work aims to explore the synchronization of music and gameplay in games, investigating techniques used in game engines and audio middleware. The motivation behind this study is the need to understand the synchronization of music and gameplay in games in order to offer an immersive experience to players. To achieve this goal, a general research was conducted on the state of the art in synchronization in games, with a focus on rhythm games as they are more demanding in this aspect. The work methodology involved the analysis of techniques used in game engines and audio libraries, as well as the definition of functional and non-functional requirements. As a result, a proof of concept of a game with real-time music and gameplay synchronization was be developed, using a specified architecture. This study contributes to the compilation and formal verification of information about music and gameplay synchronization in games, aiming to improve the understanding and application of this technique in the game industry.

Keywords: Synchronization, music, gameplay, game engines, Fmod.

Lista de ilustrações

Lista de tabelas

Lista de abreviaturas e siglas

GDD	Game Design Document
HUD	Heads-up-display
UE4	Unreal Engine 4
UI	User Interface

Sumário

1 Introdução

Os jogos com jogabilidade baseado em áudio/música se destacam como uma categoria única em comparação a outros estilos populares de jogos, como jogos de tiro, jogos de plataforma, entre outros. Enquanto em muitos jogos o áudio assume um papel secundário ao ser utilizado para feedback do jogador ou para ampliar a imersão, nos jogos com gameplay baseado em áudio/música, a música e os efeitos sonoros são parte essencial da jogabilidade. Esses jogos frequentemente apresentam uma integração criativa dos elementos sonoros e musicais, onde os jogadores interagem diretamente com a música e o áudio para progredir no jogo, resultando em uma experiência de jogo diferenciada e inovadora. Essa abordagem menos comum pode proporcionar oportunidades únicas para os desenvolvedores explorarem novas formas de integração do áudio/música ao gameplay, levando a produtos inovadores que oferecem uma experiência de jogo única e cativante para os jogadores.

1.1 Motivação

A motivação por trás deste trabalho é explorar a sincronização de música e jogabilidade. A sincronização refere-se ao processo de ajustar o tempo dos elementos do jogo, como visuais e controles, com o ritmo e o tempo da música.

A sincronização é um aspecto essencial dos jogos de ritmo, mas também desempenha um papel crucial em qualquer iteração humano-maquina como jogos de ação, aventura e esportes. Sendo importante evidenciar que há estudos sobre como a musica está conectada com a narrativa de um jogo. Por exemplo a tese *The Legend of Zelda and Leitmotif: Backtracking in an Open World*(WALCH,) em que se é explorado a musica na série de jogos *The Legend of Zelda* da Nintendo. Comprovando o impacto da musíca em jogos.

No entanto as informações técnicas sobre o assunto estão em sua maioria em blogs, fóruns ou em documentações especificas de certas ferramentas de software. Sendo assim interessante verificar ou mesmo sistematizar estas informações.

Outro ponto importante é que embora haja diversos jogos open source muitos deles utilizam a sua propria engine ou uma versão especificamente customizada (como por exemplo o jogo OSU (PPY, 2023)) e por exemplo jogos que utilizam game engines mais renomadas como por exemplo Unreal não publicam o código (exemplo Hi-Fi Rush e BPM: Bullets per minute) dificultando assim encontrar exemplos de código-fonte de projetos reais de jogos de ritmo.

1.2 Objetivo

O objetivo deste trabalho é criar uma prova de conceito de jogo envolvendo a sincronização de áudio e animação gráfica em tempo real, com uma arquitetura de software que ofereça desempenho adequado a esta tarefa. Para tanto, será necessário analisar algumas dessas técnicas utilizadas para a sincronização de áudio em uma game engine (por exemplo Unreal, Unity ou Godot) e bibliotecas para composição de áudio.

A execução de código de simulação em Game Engines como o Unity é baseada em um modelo de eventos periódicos. Entretanto, a taxa de atualização depende do desempenho do hardware, por isso há uma limitação nos instantes em que eles podem ser acionados. No caso da atualização gráfica (geração de quadros de animação) isso geralmente não é um problema, no entanto um áudio que necessite de sincronização precisa pode soar descompassado. Desta forma, um objetivo secundário é a implementação de técnicas para manter a sincronização do áudio com outros elementos de jogo.

Outro problema é que para que o áudio seja responsivo é necessário juntar múltiplos sons e embora as engines tenham os seus próprios sistemas de áudio talvez utilizar uma biblioteca ou um middleware específico de áudio como o FMOD ou Wwise seja mais vantajoso. Sendo assim importante a avaliação do uso de uma solução específica para áudio.

1.3 Justificativa

A sincronização da música com a jogabilidade é algo essencial mas muitas vezes despercebido, ele geralmente é notado quando não está funcionando, o que causa uma quebra na imersão do jogador e afetar negativamente a experiência de jogo. É fundamental que a música esteja perfeitamente alinhada com as ações do jogador para criar uma experiência de ritmo envolvente e satisfatória.

Os desafios técnicos associados à sincronização música-jogabilidade incluem a latência, que é o atraso entre a ação do jogador e a resposta sonora correspondente no jogo. A latência pode ocorrer em várias etapas do pipeline de áudio, desde a captura e processamento dos eventos de entrada do jogador até a reprodução do som. Minimizar a latência é crucial para garantir uma sincronização precisa entre a música e as ações do jogador.

Outro desafio técnico é a precisão da sincronização, ou seja, garantir que a música esteja corretamente mapeada para as ações do jogador em termos de tempo e ritmo. Para isso, é necessário considerar as variações de hardware e software dos dispositivos de reprodução de áudio, como alto-falantes, fones de ouvido e placas de som, que podem afetar a precisão da sincronização. Uma solução é a implementação de uma calibração por

parte do usuário a fim de realizar ajustes finos nas marcações de tempo para garantir que a música esteja perfeitamente alinhada com a jogabilidade.

No entanto as informações técnicas sobre o assunto estão em sua maioria dispersas em blogs, fóruns ou em documentações específicas de game engines, por isso uma compilação e verificação formal dessas informações uma contribuição para a comunidade de desenvolvedores de jogos. Oferecer uma base sólida de informações técnicas pode contribuir para o desenvolvimento de jogos de ritmo mais precisos, otimizados e envolventes, melhorando a experiência do jogador e promovendo a inovação no campo de desenvolvimento de jogos.

1.4 Organização do trabalho

Para que pudesse ser iniciado o desenvolvimento do projeto, inicialmente foi feita uma pesquisa geral sobre o estado da arte em sincronização em jogos, em especifico foram focados jogos de ritmo por serem mais exigentes neste quesito. Os principais aspectos conceituais estão descritos no capitulo 2.

No capitulo 3 é descrito a metodologia de trabalho, no capitulo 4 os requisitos funcionais e não funcionais e no capitulo 5 é detalhado o desenvolvimento.

Por fim no capítulo 6 é demonstrado o resultado final.

2 Aspectos Conceituais

Nesta seção, são abordados de forma geral os temas pesquisados para o preparo para o desenvolvimento da prova de conceito. Entre estes estão o tempo em game engines, o que são middleware (especialmente os de audio), como notas são armazenadas (timestamps, pode ser generadas em tempo real no entanto isso não será abordado), frame e fps.

2.1 Quadro e FPS

Um quadro (ou *frame*) é uma imagem dentro de uma sequência que será exibidas rapidamente em sequência para criar a ilusão de movimento contínuo. O FPS (frames per second, em inglês) é uma medida da taxa de quadros que são exibidos podem ser fixados em 30 fps ou 60 fps, mas geralmente ele dependerá do hardware.

Engines como Unreal, Unity e Godot deixam você especificar o que atualiza em cada quadro e o que atualiza no tick de simulação, que podem inclusive rodar em threads separadas. O problema é que muitas vezes a implementação pode ficar interligada com a atualização do quadro (PARSONS, 2023), restando assim duas opções. Fixar o FPS a uma taxa menor como 30 fps o que pode deixar a experiência menos fluida ou não fixar o fps e possivelmente causar uma variação na performance do jogo.

2.2 Game Engine

Uma game engine é um software que fornece uma estrutura e um conjunto de ferramentas para desenvolver jogos de forma eficiente. Ela é composta por componentes de software, como sistemas de renderização gráfica em 3D, detecção de colisões, sistemas de áudio e outros, que são separados das assets de arte, mundos de jogo e regras de jogabilidade. A separação entre os componentes de uma game engine permite que os desenvolvedores licenciem ele e o modifiquem para criar novos jogos com mudanças mínimas no software do motor. (GREGORY, 2018) Um exemplo é o Unity, um dos Game Engines mais populares e amplamente utilizados na indústria de jogos (UNITY, 2023d). Lançado em 2005 pela Unity Technologies, o Unity é conhecido por sua versatilidade, facilidade de uso, suporte multiplataforma e por ser Royalty-free além de poder ser utilizado gratuitamente para projetos com receita anual menor que 100000 dólares (UNITY, 2023d).

2.3 O tempo em game engines

A medida do tempo transcorrido no jogo é importante para diferentes funções, como animação, simulação física e execução de regras do jogo. A contagem do tempo com base no relógio do sistema é a solução mais simples, mas há outros requisitos como permitir pausas no jogo, sincronizar jogos distribuídos em rede e executar a simulação do jogo em velocidade ampliada ou reduzida, que levam à implementação de sistemas mais elaborados.

A fim de fornecer um controle de um nível de abstração maior game engines implementam formas de lidar com o tempo diferentes. No entanto cada implementação tem suas particularidades que podem trazer resultados inesperados. No caso do Unity (UNITY, 2023c) há duas formas principais de acessar a informação sobre o tempo com o jogo em execução Time.time e AudioSettings.dspTime.

O Time.time devolve o tempo do começo do quadro e pode ser alterado pelo Time.timeScale, ou seja, ele não necessariamente é o tempo real. Para contornar isso existe o Time.unscaledTime no entanto ele também é o tempo do inicio do quadro. O problema dessas implementações é que a quantidade de quadros é limitada o que gera uma discretização do tempo.

Por exemplo se um evento deve ser acionado no tempo entre dois quadros. No primeiro quadro ele não será adicionado pois seria muito cedo sendo assim ele é acionado no quadro seguinte. No entanto o segundo quadro é executado depois do momento em que o evento deveria ter sido acionado. Embora na questão visual de quadros isso não seja um problema, no contexto de sons, principalmente em musicas onde é comum ter mais do que 60BPM restringir quando um som pode ser tocado impacta na experiência do usuário.

Para obter o tempo mais próximo do real pode ser utilizado Time.realtimeSinceStartup que devolve o tempo de acordo com o tempo do sistema ou usar o AudioSettings.dspTime que devolve o tempo do sistema de áudio.

2.4 Event Dispatchers

No contexto do Unreal Engine 4 (UE4), o termo "Event Dispatcher"refere-se a um mecanismo usado para facilitar a comunicação e o manuseio de eventos em um sistema baseado em componentes. Event Dispatchers são frequentemente usados para criar sistemas de eventos que permitem que objetos notifiquem e respondam a mudanças ou ações específicas.

Pode-se fazer um paralelo entre Event Dispatchers e o padrão Publish-Subscribe reside na capacidade de vincular (ou subcrever) funções a um Event Dispatcher. Essa associação indica que tais funções devem ser acionadas quando o evento correspondente

ocorre (UNITY, 2023b), efetivamente realizando um broadcast para notificar todas as funções vinculadas sobre a ocorrência do evento.

Assim como no padrão Publish-Subscribe, onde os observadores (ou subscribers) se registram para receber notificações de um tópico específico, as funções vinculadas a um Event Dispatcher atuam como observadores do evento em questão. Dessa forma, esse mecanismo proporciona uma maneira eficaz de implementar a comunicação desacoplada entre componentes, uma característica fundamental do padrão Publish-Subscribe.

Em jogos de ritmo, a utilização de Event Dispatchers torna-se uma ferramenta crucial no contexto da sincronização da música com a jogabilidade. Ao lidar com a disseminação de informações entre objetos, essa abordagem oferece uma solução elegante para gerenciar eventos de som específicos. Em muitos casos, a informação sobre quando um evento de som deve ser tocado é centralizada em um único objeto. Para evitar a criação de dependências extensas com todos os objetos envolvidos na execução de ações relacionadas ao som, é mais prático e modular que cada objeto implemente internamente como deve reagir a determinado evento. Isso contrasta com a abordagem de consolidar todas essas informações em um controlador massivo, promovendo um design mais flexível e modular para o desenvolvimento de jogos de ritmo onde a grande maioria dos objetos devem agir de forma sincronizada.

2.5 FMOD

O FMOD (LTD, 2023) é um middleware, ou seja, um software que atua como um intermediário entre diferentes sistemas, facilitando a comunicação e a integração entre eles. Ele proporciona uma camada de abstração que permite a interoperabilidade entre sistemas heterogêneos. No contexto específico deste trabalho, é utilizado um middleware de áudio, o FMOD, que se destaca por ser uma ferramenta especializada, projetada para lidar com aspectos específicos relacionados ao áudio em jogos.

Vale ressaltar que ele é diferente de uma biblioteca, que são conjuntos de código pré-compilado que fornecem funcionalidades específicas para serem utilizadas por um programa. Elas são geralmente mais específicas e diretas em comparação com middleware. Enquanto o middleware age como um intermediário entre sistemas tendo uma interação mais similar a uma API, as bibliotecas são incorporadas diretamente no código de um programa para fornecer funcionalidades específicas.

O FMOD Studio apresenta uma abordagem única na criação e manipulação de áudio através de uma estrutura chamada Event. Os usuários utilizam a interface intuitiva do FMOD Studio para desenvolver e ajustar esses eventos, que podem ser posteriormente exportados como Event Banks. Estes Event Banks são essencialmente formatos binários que podem ser carregados no jogo através de um plug-in 1 disponibilizado no site do

FMOD (LTD, 2023), integrando perfeitamente a experiência sonora ao contexto do jogo.

Figura 1 – Tela de plug-ins do Unreal

Cada evento dentro do FMOD Studio é organizado em uma linha do tempo 2 que compreende várias faixas, sendo uma delas a faixa Master. Essa estrutura de linha do tempo permite a colocação de módulos de som em suas respectivas faixas, determinando quando cada módulo será acionado durante a execução do evento. A linha do tempo é essencial para a sincronização precisa de efeitos sonoros, músicas e outros elementos auditivos. (MIZUTANI,)

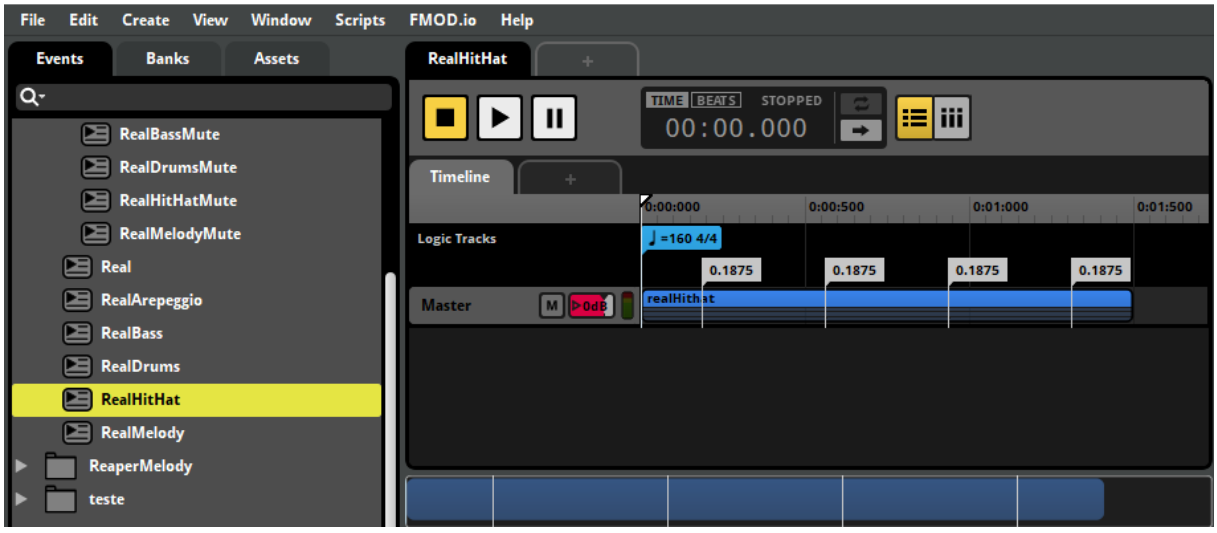


Figura 2 – Linha do tempo no FMOD Studio com somente a faixa master

Além disso também há os parâmetros cuja principal função no FMOD é permitir que o código do jogo atualize dinamicamente os valores associados a propriedades de eventos. Esses valores podem ser usados para automatizar propriedades de eventos, controlar a linha do tempo do evento por meio de marcadores lógicos e acionar instrumentos nas folhas de parâmetros do evento. Esse exemplo 3 mostra o paramêtro RealMelodyParam que controla o volume da track Melody.

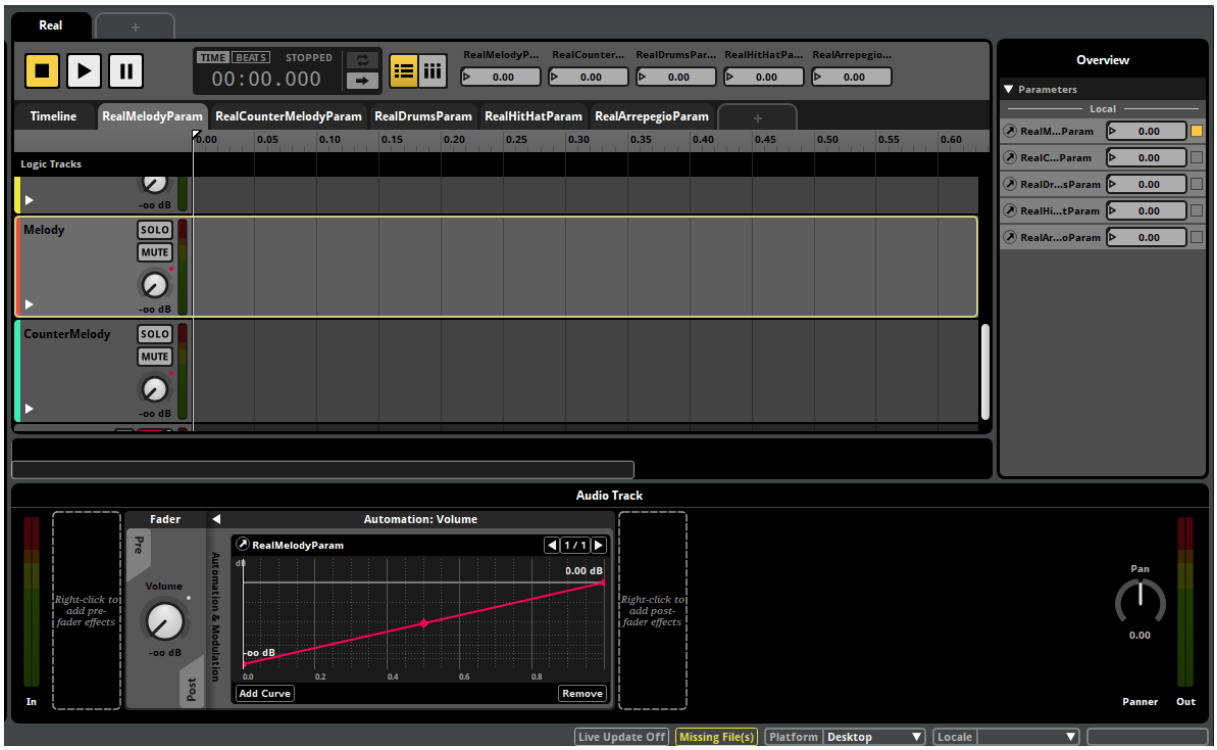


Figura 3 – Parâmetro RealMelodyParam no FMOD Studio

Por fim *Timeline Markers* (marcadores de linha do tempo) no contexto do FMOD Studio são pontos específicos ao longo da linha do tempo de um evento sonoro. Eles são usados para marcar momentos cruciais ou pontos de referência no desenvolvimento do evento, proporcionando uma maneira organizada de estruturar e controlar o fluxo temporal do áudio. No caso para esse projeto eles são utilizados somente para utilizar o evento *OnTimelineMarker* implementado pelo Fmod para enviar infomações sobre a posição do áudio para o Unreal. Eles podem ser vistos na figura 2 pelos retângulos nomeados 0.1875.

2.6 Musica e gameplay

A música em jogos, também conhecida como trilha sonora, desempenha um papel na ambientação do jogo. Por isso é interessante modificar a música de acordo com eventos ou contextos específicos do jogo. Por exemplo, em momentos de ação intensa, a música pode se tornar mais acelerada e emocionante para aumentar a adrenalina dos jogadores. Em contrapartida, em momentos de calma ou suspense, a música pode se tornar mais

suave e sutil, contribuindo para a imersão na narrativa do jogo. No artigo publicado pelo AWE Interactive (INTERACTIVE, 2023) é especificado 3 formas em que a gameplay pode interagir com o áudio:

- Note Charts: Todo áudio tem metadados que informam quais eventos devem ser acionados
- Musica Procedural: O áudio é feito de segmentos que são escolhidos por algum algorítimo, sendo que cada segmento tem eventos especificos
- Somente a batida: No caso somente a batida é relevante e os eventos são sinconizados com ela.

2.7 Classe Condutor

O conceito da classe condutor(FIZZD, 2020) é uma classe utilizada para manter o ritmo e a posição do áudio. Todo o sistema deve girar em torno desta classe.

O propósito principal da classe Conductor é estabelecer um ponto de referência contínuo para o andamento da música, garantindo que o ritmo do jogo permaneça consistente, independentemente das variações na taxa de quadros (fps).

Para desacoplar o conceito de tempo do frame a classe Conductor utiliza a posição do áudio como um ponto de referência constante. Essa posição é incrementada de forma consistente, independentemente da variação de fps, garantindo que a posição do áudio, nesse contexto, funciona como um marcador temporal.

Ao evitar a redefinição da posição do áudio a cada quadro, a classe Conductor previne inconsistências e imprecisões na sincronização. Se dependêssemos da redefinição constante do ponto de referência a cada frame, a variação no fps poderia resultar em eventos acontecendo fora do ritmo desejado. Incrementando a posição do áudio de forma fixa, a classe Conductor supera esse desafio e proporciona uma experiência mais fluida e coesa.

Um ponto importante é que a classe condutor também está presente em outros projetos como Friday Night Funkin(PRINCESSMTH, 2021)

Algumas nomenclaturas importantes a serem utilizadas definidas em (FIZZD, 2020) são:

- Songposition: é uma variável que deve ser configurada diretamente a partir da variável correspondente no objeto de áudio.
- Crotchet: se refere a uma semínima na notação musica e depende do valor do BPM - Batidas Por Minuto de um musica.

crotchet, which gives the time duration of a beat, calculated from the bpm

2.8 Game Design Document

O Game Design Document (GDD) é um documento utilizado na fase de pré-produção do desenvolvimento de jogos de vídeo, que serve como base para o design, desenvolvimento e validação do software durante a fase de produção. Ele é utilizado para estabelecer a estrutura, detalhes e relacionamentos entre os elementos do jogo, visando aprimorar a experiência do jogador durante o tempo de jogo. (GONZáLEZ-SALAZAR et al., 2012)

Não existe um modelo especifico para um GDD uma vez que eles podem variam em complexidade e detalhamento com base na natureza do projeto. Alguns GDDs podem ser mais visuais, incorporando conceitos de design, esboços e mapas, enquanto outros são mais textuais, detalhando informações de forma mais técnica. A escolha do modelo geralmente depende do tipo de jogo, das preferências da equipe de desenvolvimento e da necessidade de comunicar efetivamente as ideias do jogo.

No entanto vale notar que há um similaridades entre o GDD e documentos de requisitos em engenharia de software. Ambos servem como documentos mestres que definem o escopo, requisitos e metas do projeto. Podendo fazer a seguinte comparação (GONZáLEZ-SALAZAR et al., 2012):

1. **História e Jogos de referencia:** Detalhes sobre personagens, enredo e narrativa, delineando o universo do jogo e as interações que ocorrerão e o contexto disso no mundo real. Pode ser entendido como a relação com outros documentos e a definição do vocabulário a ser utilizado.

2. **Gameplay e Mecânicas de Jogo:** Descrição detalhada das principais mecânicas, sistemas de jogo e como eles interagem para criar a experiência desejada. Pode ser entendida como a organização dos requisitos do jogo.

3 Metodologia do Trabalho

A metodologia do trabalho é composta de seis fases: Pesquisa detalhada, Elaboração do Game Design Document (GDD), Especificação de requisitos, Escolha de Tecnologias, Definição de métricas de avaliação e testes, e por fim o Desenvolvimento do Jogo.

3.1 Pesquisa detalhada

Inicialmente, foi feita uma pesquisa mais detalhada sobre como game engines funcionam para a escolha das tecnologias utilizadas. No caso mais especificamente de como o tempo é tratado nelas e como middlewares de áudio funcionam. Além disso foi estabelecido a forma que a gameplay e o áudio serão interligadas.

3.2 Elaboração do Game Design Document (GDD)

Elaboração do documento de design do jogo (GDD), disponibilizado no apêndice A, que descreve detalhadamente todos os aspectos do jogo, incluindo a história, personagens, mecânicas de jogo, elementos de gameplay, interface do usuário, fluxo de jogo e outros elementos relevantes. O GDD será uma referência para o desenvolvimento do jogo, garantindo a compreensão clara dos requisitos do projeto.

3.3 Especificação de requisitos

Com base na revisão bibliográfica e no GDD foi feita a especificação de requisitos a fim de estabelecer claramente quais features serão implementadas no protótipo do jogo. Foi incluído nessa fase a descrição da estrutura de classes e arquitetura do projeto por meio da criação de diagramas e documentação adequada para guiar a implementação do protótipo.

3.4 Escolha de Tecnologias

Selecionar as tecnologias específicas que serão utilizadas para a implementação do jogo, levando em consideração o GDD. Escolher uma game engine que suporte a utilização dos padrões desejados, selecionar bibliotecas apropriadas que facilitem a implementação e escolher o middleware de áudio utilizado. No primeiro momento foi adotada a game engine Unreal Engine que tem integração com o middleware de áudio FMOD.

3.5 Definição de métricas de avaliação e Testes

Após a conclusão da implementação do jogo foi realizado avaliações e testes. Para avaliar a sincronização de ataques com a música, foram definidas as seguintes métricas:

Precisão: Refere-se à proximidade dos ataques em relação ao ponto de referência contínuo definido pela classe conductor, implementado como o *Timeline Position*. Quanto mais próxima de zero for a diferença entre o *Timeline Position* e a posição atual da nota , maior é a precisão.

Acurácia: Representa a consistência da sincronização ao longo do tempo. Uma alta acurácia indica que o sistema mantém a sincronização, diferença entre o *Timeline Position* e a posição atual da nota de maneira consistente, independentemente das variações temporais. Ela será calculada como a variação a diferença entre o *Timeline Position* e a posição atual da nota

Além disso será avaliado a diferença entre performance da *Timeline Position* com a função implementada da Unreal Engine *GetTimeSeconds*.

3.6 Desenvolvimento do Jogo

O cronograma planejado para as etapas de desenvolvimento do jogo inclui a criação de personagens e ambiente, implementação de inimigos, comando de soldados e fortalecimento, menu inicial e configurações. A definição das datas levou em conta um desenvolvimento mais lento entre maio e agosto devido à carga horária menor nesses meses.

1. **01/05/2023 - 28/05/2023: Desenvolvimento de personagens e ambiente**
    - Arranjar os modelos e animações dos soldados, com movimentação suave e responsiva.
    - Criar o ambiente do jogo em estilo isométrico, com colisões.
    - Implementar a mecânica de movimentação do personagem e a lógica de colisão.
2. **29/05/2023 - 23/09/2023: Inimigos**
    - Encontrar placeholders dos modelos e se possível animações dos inimigos, com padrões de movimentação e comportamento.
    - Implementar a lógica de ataque dos inimigos.
    - Integração do FMOD para controlar os eventos sonoros relacionados aos inimigos, principalmente o ritmo.
3. **24/09/2023 - 30/10/2023: Comando de soldados e fortalecimento**

- Implementar a mecânica de seleção e comando dos soldados, permitindo que o jogador selecione alvos e os comande em combate.
- Implementar a mecânica de fortalecimento dos soldados, permitindo que o jogador aumente o dano deles de acordo com o ritmo do inimigo selecionado.
- Implementar a calibração da entrada do jogador para o ataque.
- Integração do FMOD para controlar os eventos sonoros relacionados ao ataque fortalecido.

4. **01/11/2023 - 05/11/2023: Menu inicial e configurações**

- Definição do fluxo de navegação do menu.
- Implementação das telas do menu principal.
- Implementação de funcionalidades do menu, como iniciar novo jogo, carregar jogo, opções e sair do jogo.
- Implementação de configurações do jogo, como ajustes de áudio e controles.

Fase	**Maio**	**Junho**	**Julho**	**Agosto**	**Setembro**	**Outubro**	**Novembro**
Desenvolvimento de personagens e ambiente	X	X	X	X			
Inimigos					X	X	X
Comando de soldados e fortalecimento						X	X
Menu inicial e configurações							X

Tabela 1 – Linha do tempo do desenvolvimento do jogo (em meses)

4 Especificação de Requisitos

Com base no Game Design Document (GDD) A desenvolvido foi definido os requisitos funcionais e não funcionais.

4.1 Requisitos Funcionais

Os requisitos funcionais mínimos esperados são:

- Comando de soldados: permitir que o jogador comande seus soldados, selecionando alvos para atacar no modo de combate. Deve ser possível selecionar mais de um ao mesmo tempo. Caso o comando de movimentação seja para uma area restrita ele deve ser cancelado
- Fortalecer os soldados: O jogador deve ser capaz de aumentar o dano de seus soldados clicando no mesmo ritmo que o inimigo selecionado.
- Menu principal: permitir que o jogador acesse o menu principal para acessar as opções de jogo, como ir para a próxima missão, mexer nas configurações e sair do jogo.
- Inimigos: Os inimigos devem atacar os soldados baseado em um padrão musical
- Calibração: O jogo deve permitir que o jogador calibre o áudio com a gameplay, de forma que a música e os efeitos sonoros estejam sincronizados com as ações dos personagens. Essa calibração deve ser possível em diferentes tipos de hardware, para garantir que o jogo possa ser jogado em diversos computadores e dispositivos.
- Visualização: No modo de exploração e combate ele sera de forma isométrica. No menu principal sera um angulo frontal de uma sala. Um exemplo pode ser visto no Game Design Document.

4.2 Requisitos Não Funcionais

- Interface de usuário: Ter uma interface que utilize assets além dos disponibilizados no tutorial das game engines
- Salvar os dados do usuário: Ser possível salvar e retomar o progresso.

- História: O jogo deve ter uma história que envolva o jogador e motive-o a avançar nas missões. Essa história será feita principalmente pelos colecionaveis uma vez que diálogos, cutscenes, eventos e outros elementos narrativos podem ser tornar muito custosos.
- Performance: O áudio e a gameplay devem estar sincronizadas de acordo com as métricas definidas. falar mais tecnicamente canais de audio, eventos,
- Feedback: O jogo deve dar respostas claras de acordo com a performance do usuário, falar como que o middleware deve agir para dar o feedback
- Colecionáveis: permitir que o jogador encontre colecionáveis escondidos que forneçam informações sobre a história do jogo.
- Descarte de colecionáveis: permitir que o jogador descarte colecionáveis no menu principal, reduzindo seu valor pela metade.
- Configurações: Ser possível alterar o som e a vinculação das teclas
- Tecnologia: integrar com o FMOD de modo a ter mais controle do áudio.

4.3 Arquitetura do projeto

Tendo como base os requisitos expostos anteriormente, foi desenvolvida uma arquitetura para o jogo, representada de maneira abstrata no diagrama de classes apresentado na Figura 4. É crucial notar que essa representação visual de objetos isolados, como personagens, inimigos, itens e a Câmera, é uma simplificação destinada a facilitar a compreensão da estrutura do jogo.

No contexto de um diagrama de classes, a câmera pode ser representada como uma classe que contém propriedades e métodos relacionados à sua funcionalidade, como a posição, a rotação, o campo de visão, etc. Essa representação em um diagrama de classes pode ajudar a compreender a estrutura e o funcionamento da câmera como um componente isolado do jogo.

No entanto, na realidade, a câmera não existe de forma isolada, mas sim como parte integrante da game engine. Durante a execução do jogo, a câmera é conectada à game engine e é usada para visualizar a cena do jogo a partir de uma determinada perspectiva. A game engine é responsável por atualizar a posição, a rotação e outros aspectos da câmera com base na lógica do jogo e nas ações do jogador.

Dessa forma, os objetos que estão representados como classes em um diagrama de classes estão, na verdade, conectados e interagindo entre si na game engine, mesmo que não estejam visualmente representados dessa forma no diagrama de classes. A game

engine é responsável por gerenciar a posição, movimentação, interação e outros aspectos desses objetos, tornando-os componentes integrados da experiência de jogo.

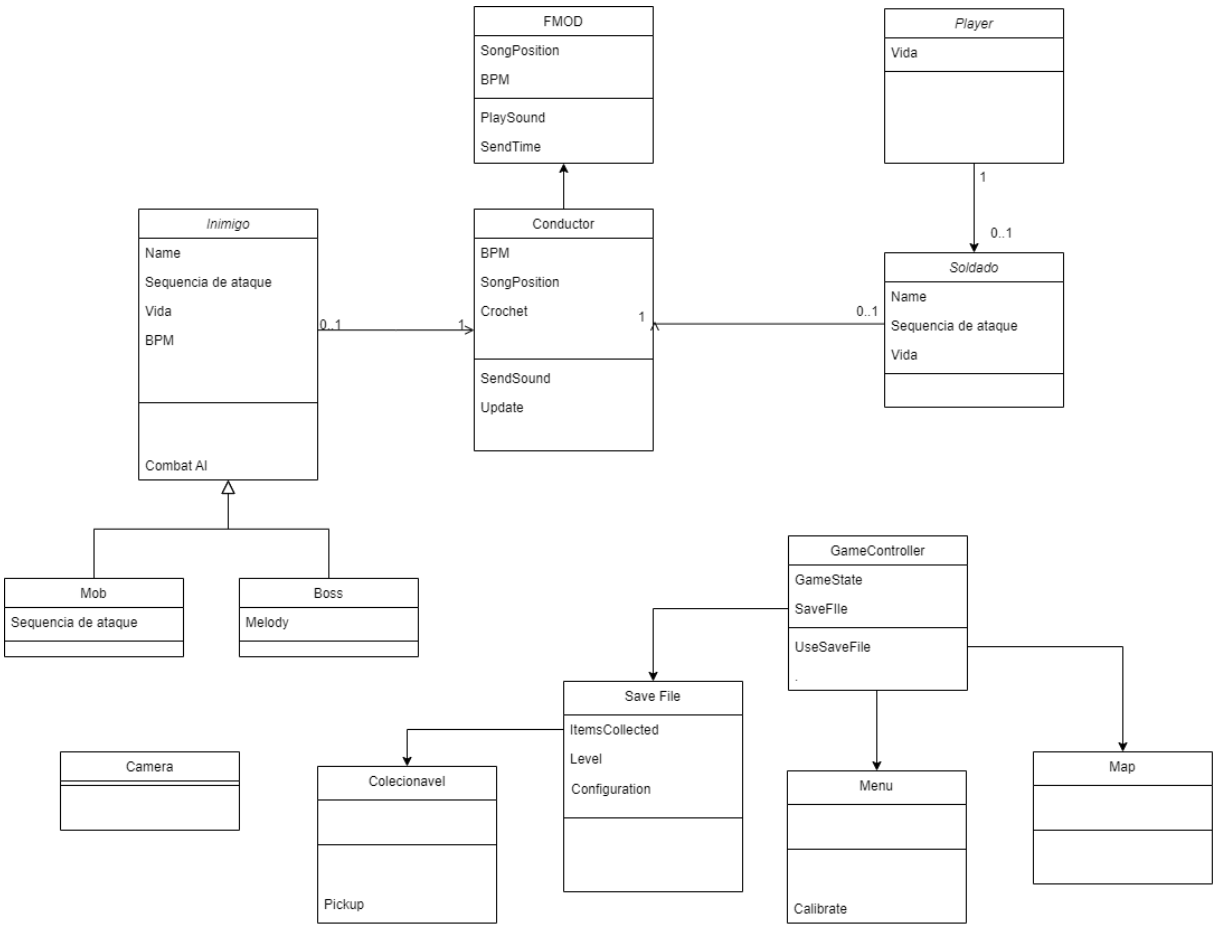


Figura 4 – Diagrama de Classe

Além da classe Condutor definida anteriormente 2.7 as principais classes são a FMOD, Inimigo, Player e GameControler.

A classe "FMOD"é a Classe FMODAudioComponent implementada pelo FMOD para o Unreal interagir com ele permitindo a reprodução de eventos sonoros do FMOD Studio, oferecendo controle dinâmico em tempo real, suporte para Blueprints, notificações de eventos e configurações de parâmetros.

A classe "Inimigo"é a base que representa os inimigos que possuem características como nome e vida, e podem interagir com outros elementos do jogo. Sendo que o foco principal dela é ter uma sequência de ataque sincronizada com um ritmo ou melodia.

Já a classe "Player"representa como o jogador pode interagir com o ambiente do jogo, combater inimigos, coletar itens e avançar na história ou objetivos do jogo ao controlar diversos Soldados. O Player possui somente a vida que é um atributo que indica a quantidade de recursos disponíveis ao jogador. A perda total da vida é a unica forma de perder o jogo.

A classe "GameController"é responsável por gerenciar a lógica central do jogo. Ela controla eventos, fluxo de jogo, recursos, detecção de colisões, atualização do estado do jogo e comunicação entre diferentes elementos do jogo, como o jogador, os inimigos e outros objetos do jogo.

5 Desenvolvimento

5.1 Escolha da Game Engine

A escolha da game engine para o desenvolvimento de um projeto é uma decisão crucial que impacta diretamente no processo de criação e no resultado final do jogo. Diversos fatores foram considerados para tomar essa decisão, levando em consideração as necessidades específicas do projeto em questão.

5.1.1 Visualização 2D ou 3D

A primeira consideração foi determinar se o jogo deveria ser desenvolvido em 2D ou 3D. O projeto poderia ser realizado em ambas formas. Sendo assim o que teve um peso maior foi a disponibilidade de assets (Recursos que compõem um jogo, como modelos 3D, texturas, animações, músicas, efeitos sonoros, etc. No caso, está sendo considerado somente recursos visuais) gratuitos uma vez que a criação de assets não é o foco central do projeto.

O Unreal Marketplace (ENGINE, 2023b), loja virtual para assets criados para o Unreal Engine, oferece mensalmente pacotes de assets gratuitos, proporcionando uma fonte valiosa de recursos visuais. A autora já possuía o pacote "Isometric World : Sky Temple"(GAMEASSETFACTORY, 2023) enquanto que o Unity apesar de ter a sua própria asset store com assets gratuitos eles não são tão extensivos como os do Unreal.

5.1.2 Game Engine com visualização 3D

Tanto o Unity quanto o Unreal Engine tem suporte para a criação de jogos tridimensionais. O Unreal é conhecido por ter gráficos mais fotorrealistas do que o Unity enquanto que o Unity é conhecido por conseguir otimizar jogos para dispositívos mais fracos como celulares(JOHNS, 2023).

Vale notar que a autora tem experiência prévia com o Unreal Engine e isso foi um fator que influenciou positivamente a escolha final da plataforma de desenvolvimento.

5.1.3 FMOD

A escolha da engine também envolveu considerações relacionadas à integração com o FMOD. A documentação do FMOD para Unity é mais detalhada que a para Unreal, fornecendo exemplos de script e API que facilitaram a integração de áudio de forma mais eficiente. No entato a integração com o Unreal Engine é igualmente funcional.

Considerando que dificuldades na documentação do FMOD podem ser resolvidas mais facilmente do que a criação de assets foi escolhido o Unreal Engine.

5.2 Configuracao do FMOD

Nesta seção, detalhamos a configuração do projeto no FMOD, o middleware escolhido para a implementação do sistema de áudio no jogo. Abordaremos em detalhes a criação de eventos e automações, além de explicar a integração essencial com o Unreal Engine, permitindo ajustes dinâmicos de volume em tempo real.

Informações detalhadas das classes FmodController e FmodControllerMelody, implementadas para a distribuição eficiente de informações entre o FMOD e o Unreal Engine, serão detalhadas na próxima seção 5.3.

5.2.1 Projeto no Fmod

O projeto foi dividido em dois tipos de eventos. Um evento com todos os instrumentos e eventos somente com os timemarkers.

O evento nomeado real com todos os instrumentos, como pode ser visto na imagem 5, serve para produzir os sons.

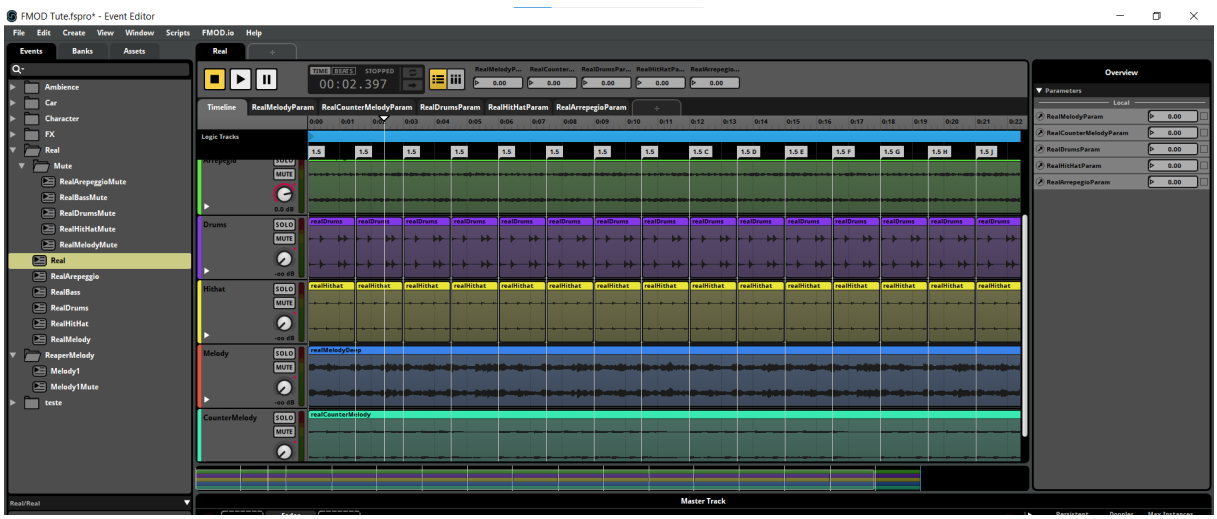


Figura 5 – Tela do projeto no Fmod

Já os eventos com dentro da pasta "Mute"tem somente os timemarkers 6, sendo responsáveis por informar ao Unreal Engine quando cada nota do instrumento é tocada.

5.2.1.1 Transformacao de Midi em timemarks

Para adicionar os timemarks nos arquivos no Fmod foi adaptado o código do projeto fmod-midi-to-markers (JEREMYABEL, 2023). O código foi ajustado para seguir um BPM

de 160 e nomear os "markers"de acordo com o tempo em que a nota é tocada.

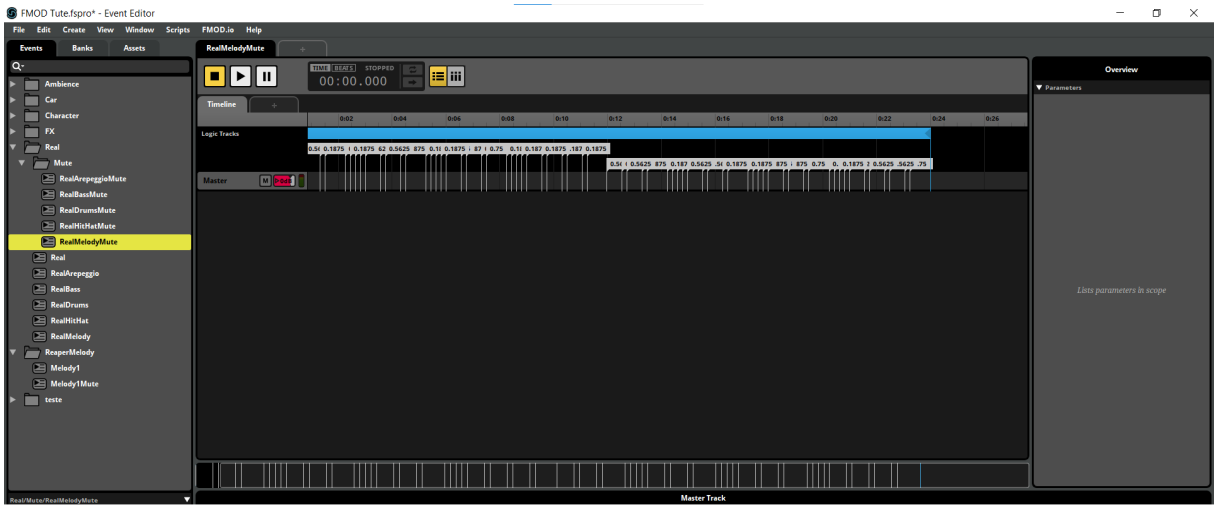


Figura 6 – Exemplo de faixa com TimeMarks

5.2.2 Integração com o Unreal

Para fazer a integração foi adicionada parâmetros para cada instrumento no evento com todos os instrumentos 5. Cada parâmetro está associado a um instrumento e permite que o Unreal Engine altere dinamicamente o volume do som em tempo real.

Para evitar dessincronizar os instrumentos eles nunca são individualmente parados, somente eh configurado o som para 0.

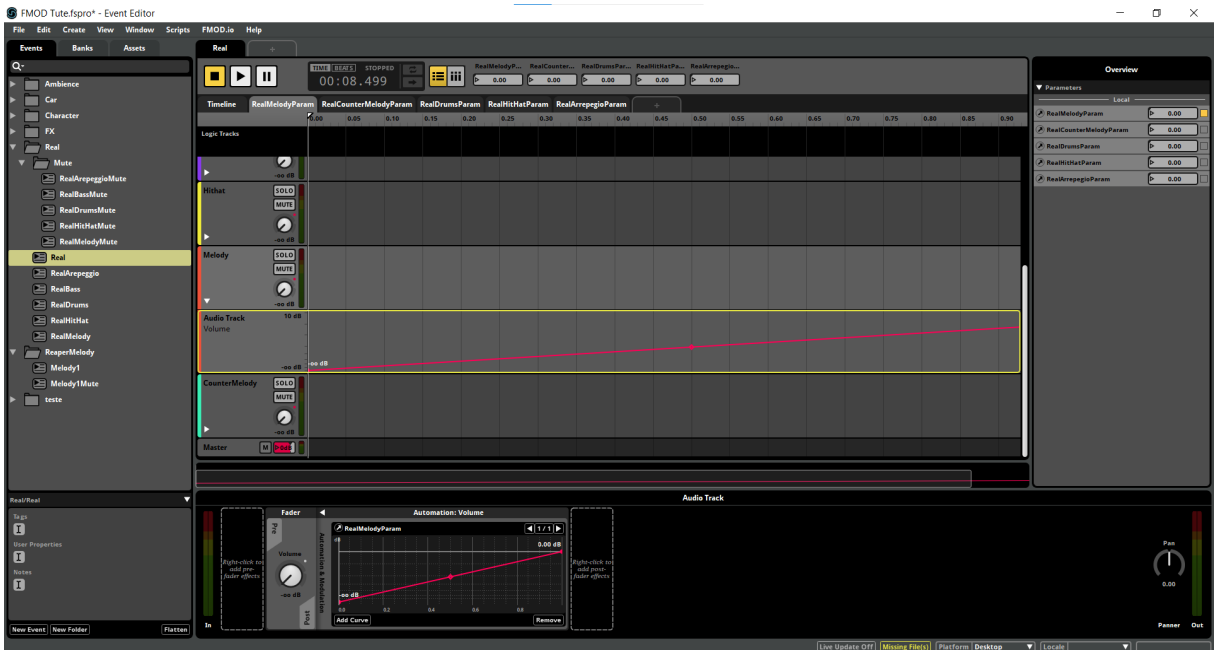


Figura 7 – Parametro no Fmod

No Unreal Engine, as informações do FMOD são recebidas através do componente FmodAudio Component, implementado pelo FMOD. Utilizando "dispatchers", essas informações são distribuídas pela classe criada FmodController, sendo a classe FmodControllerMelody responsável pelo controle dos parâmetros.

Uma analogia útil é que o FmodController lida com os eventos sem som no FMOD, enquanto o FmodControllerMelody controla o evento com os instrumentos

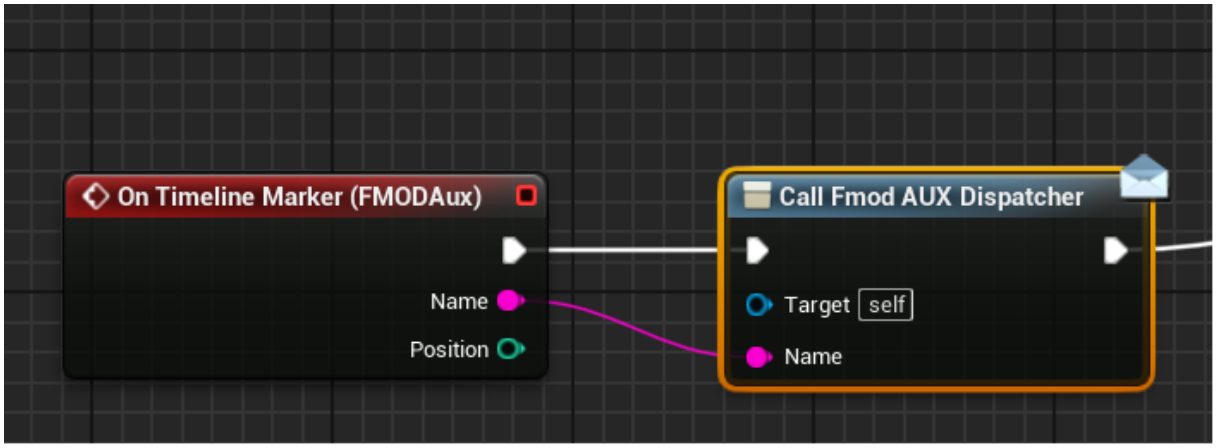


Figura 8 – Dispatcher no FModController

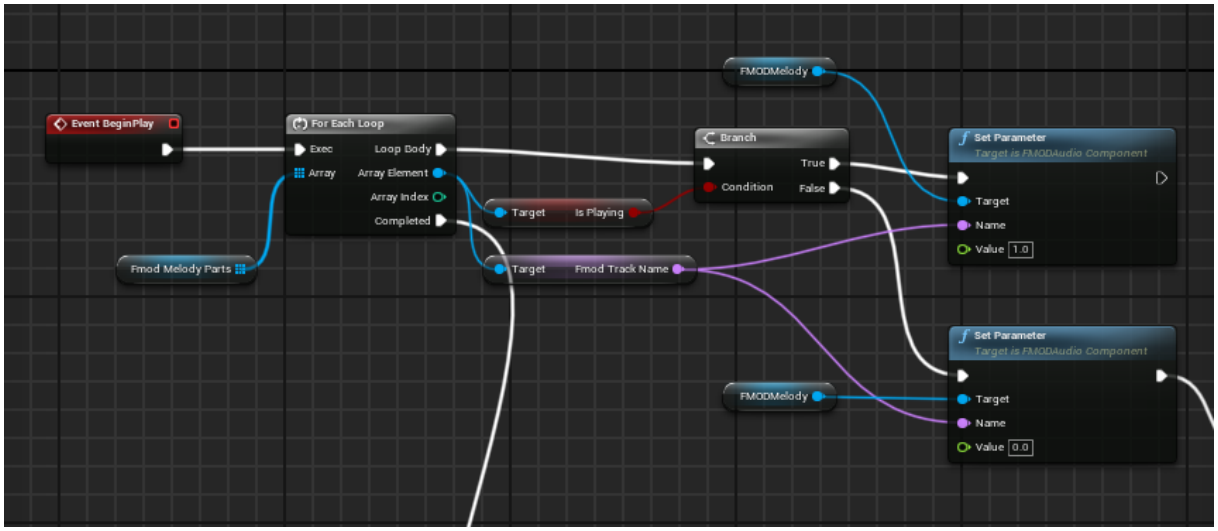


Figura 9 – Configuracao dos parametros do Fmod no FmodControllerMelody

5.3 Configuracao no Unreal

Nesta seção, abordamos a configuração do sistema no Unreal Engine, enfocando as classes e estruturas fundamentais para o sistema. Destacamos a implementação das classes FmodController e FmodControllerMelody, responsáveis por gerenciar a distribuição eficiente de informações entre os eventos do FMOD e os elementos interativos do jogo.

Exploraremos a configuração das classes BaseSoldier e BaseEnemy, representando os objetos controlados pelo jogador e os inimigos, respectivamente. Destacaremos como essas classes interagem com o sistema de áudio, proporcionando uma experiência envolvente e dinâmica para o jogador.

Por fim, discutiremos a implementação da interface do usuário (UI) e do Head-Up Display (HUD), elementos cruciais para a experiência visual e interativa do jogador durante a partida.

5.3.1 Classe Fmod e Conductor

Durante o processo de desenvolvimento, foi constatado que, devido à existência de múltiplas tracks, uma para cada inimigo, seria mais apropriado criar um condutor individual para cada tracks, em vez de manter todas as trilhas agrupadas conforme inicialmente planejado.

Dessa forma a classe condutor 12 foi dividida em duas classes a classe FmodController sendo responsável por definir os intervalos a serem incrementados de acordo com as informações das notas do Fmod e a classe FMODMelodyController é responsável por definir a posição do áudio além de consolidar todos os objetos FmodController e gerenciar a reprodução dos áudios correspondentes.

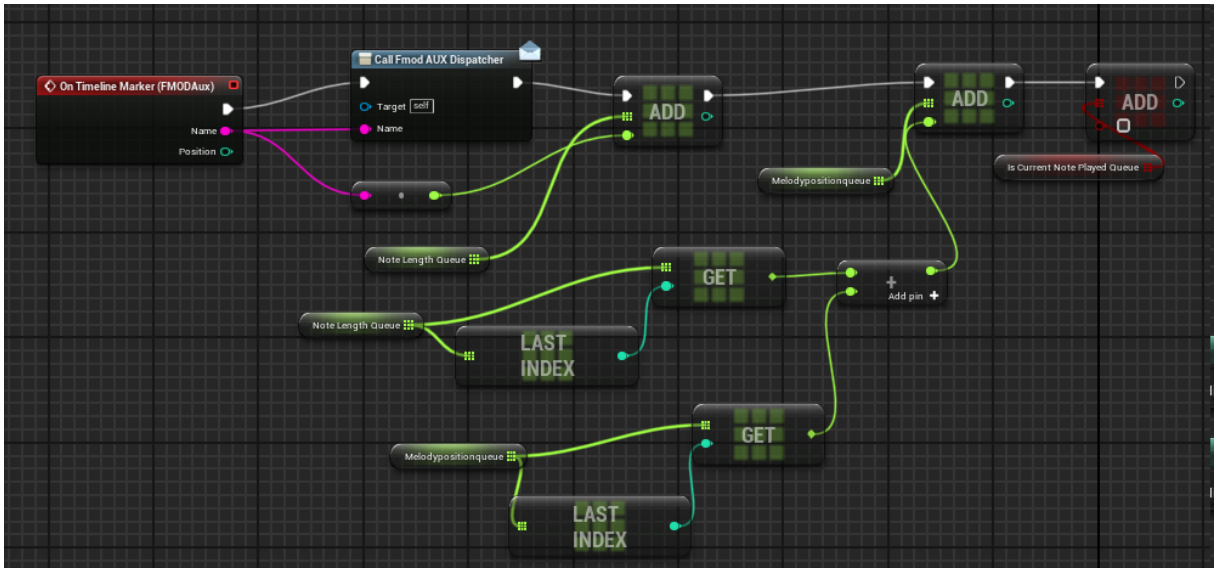


Figura 10 – "Crochet"implementado no FmodController

Além disso foi feita uma modificação no FmodController 10. Em vez dela fazer incrementos constantes baseado em crochets como indicado em (FIZZD, 2020). Ela faz incrementos baseado no comprimento de cada nota, ou seja, quantidades em função do crochet. E ela guarda uma fila das notas a serem tocadas (*MelodyPositionQueue*). Essas mudanças foram realizadas para poder adicionar um sistema de tolerância tanto antes

quanto depois de uma nota ser tocada, há dois textitFmodAudioComponents para que o Fmod possa mandar pelo FmodAux informações sobre a futuras notas antes do som ser realmente tocado. O FmodMarker sinaliza quando a nota deve ser retirada da fila de notas a serem tocadas.

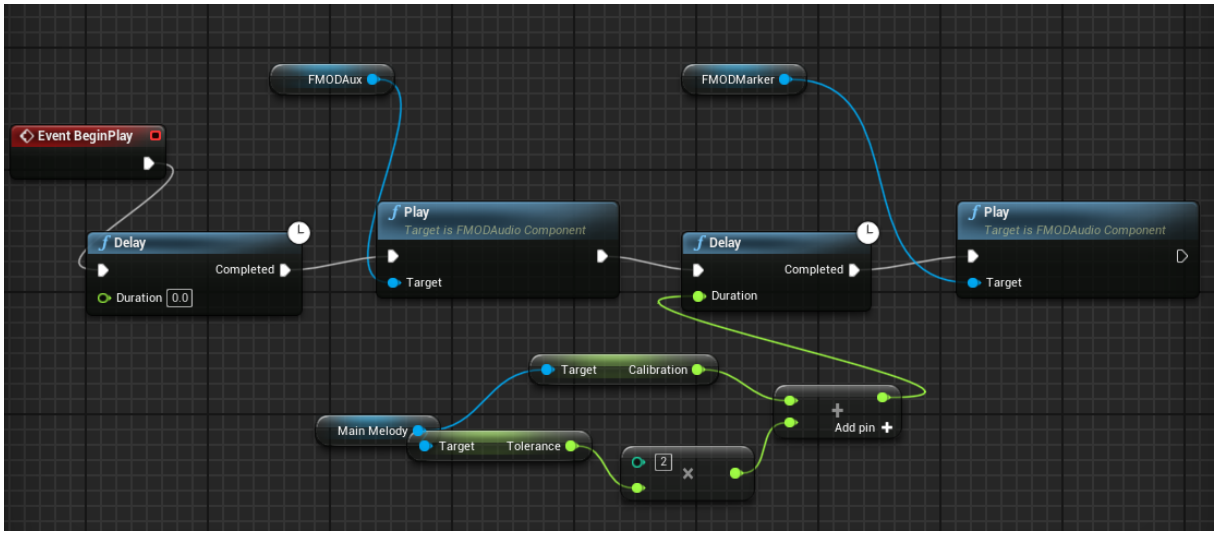


Figura 11 – Dois *FmodAudioComponent* para o sistema de tolerância

As classes BaseSoldier e BaseEnemy fazem referência apenas ao FmodController uma vez que é pressuposto que cada soldado ou inimigo deve estar associado a apenas uma trilha específica em dado momento.

Além disso foi na classe FmodController que foi implementado o sistema de pontuação do ataque dos soldados em vez na classe do soldado em si. Essa escolha foi tomada em favor de uma concentração maior do acesso a classe do FMOD uma vez que múltiplos soldados poderiam estar atacando somente um inimigo. Sendo assim desnecessário fazer o mesmo calculo múltiplas vezes.

5.3.2 Classe Soldado

São os objetos que podem ser controlados pelo jogador. Uma captura de tela da implementação pode ser vista na figura 16

É a classe responsável por dar dano aos inimigos com base na pontuação de ataque fornecido pelo FMOD Controller.

Além disso ela é a classe que o sistema de targeting se encontra

5.3.2.1 NiagaraTarget

A classe BeanTargetSystem que pode ser vista na imagem 17 foi implementada para o sistema de targeting. Ela utilizada o efeito de feixe (beam) com o sistema de partículas

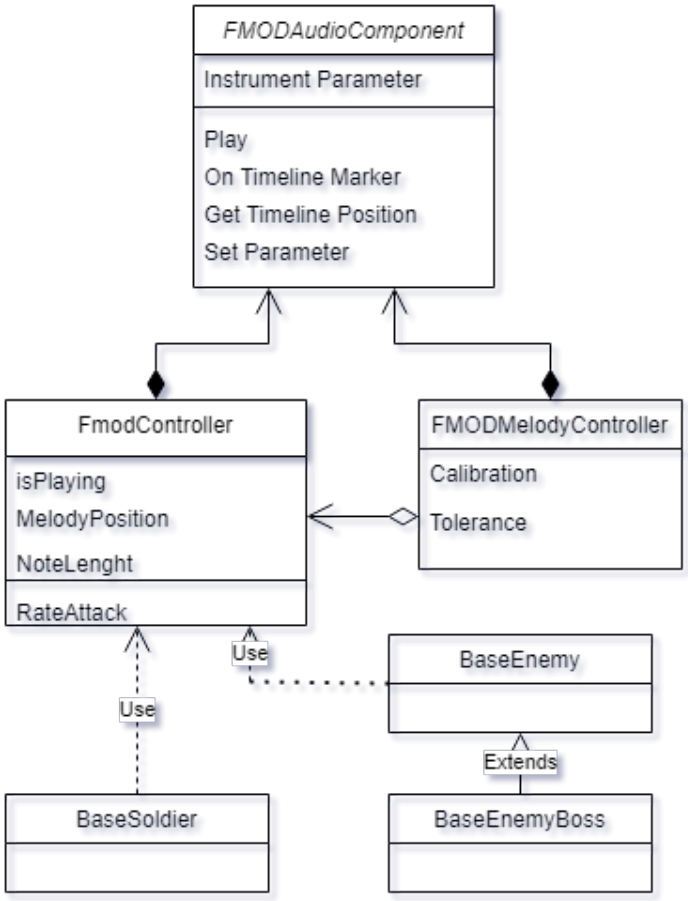


Figura 12 – Diagrama de Classe do tratamento do FMOD

Niagara no Unreal Engine. (UNITY, 2023a) A classe Soldado controla quando o efeito deve ser ativado ou não e a posição final desejada.

5.3.3 Classe Inimigo

São os objetos que atacam os soldados, foi configurado para que a animação de ataque seja em função do FmodController 13.

A partir dela é possível criar variações da classe inimigo com diferentes meshes, valores de vida, melodias, etc. É possivel observar a diferença da classe BaseEnemy 19 com a classe BaseEnemyBoss 20

5.3.4 RTS_PlayerController

A classe RTS_PlayerController extende a classe base APlayerController do Unreal Engine ela é a responsável pelo:

- Controle de Câmera: Gerenciamento da posição e zoom da câmera para proporcionar uma visão abrangente do campo de jogo.

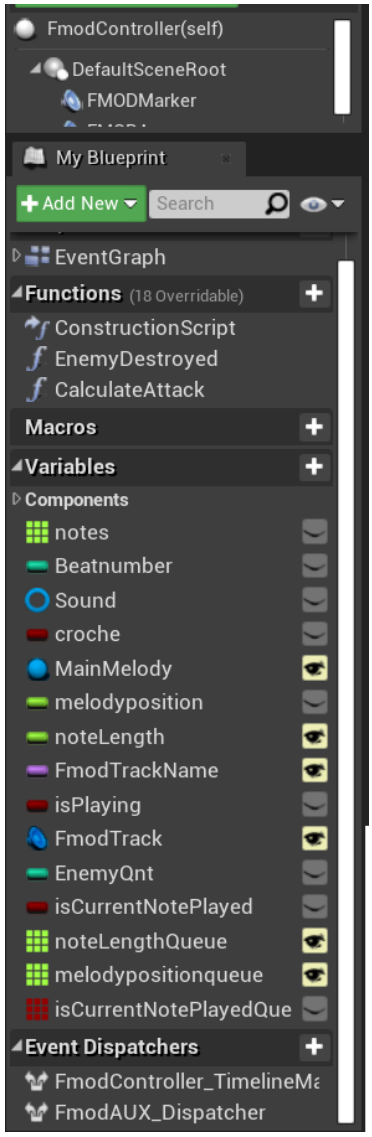


Figura 13 – FmodController

- Seleção de Unidades: Implementação de mecanismos para selecionar e controlar unidades individuais ou em grupo.
- Emissão de Comandos: Captura de input do jogador para emitir comandos às unidades selecionadas
- Interação com o HUD: Atualização e exibição de informações relevantes no HUD

5.3.5 UI e HUD

Para implementar a User Interface (UI) foi necessário utilizar duas abordagens diferentes. Uma utilizando a classe HUD (Heads-up Displays) para exibição de elementos com base na posição do cursor e a outra por meio de Widgets, elementos criados por meio do *Unreal Motion Graphics* (UMG).

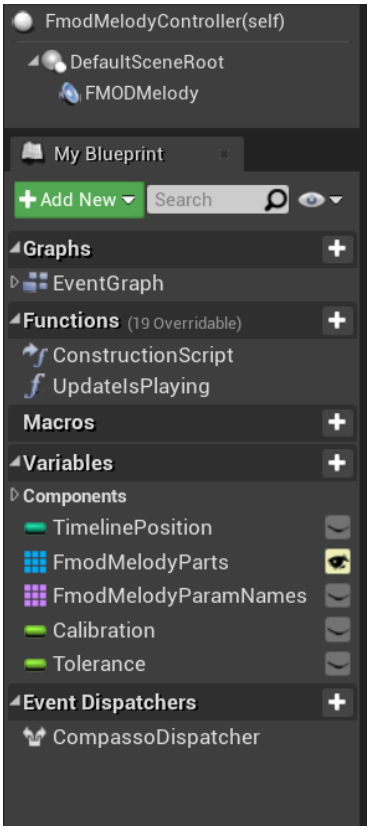


Figura 14 – FmodControllerMelody

5.3.5.1 HUD

A classe HUD(Heads-up Displays) é classe base que implementa os elementos na tela no Unreal Engine. Para realizar a parte de selecionar múltiplos soldados 21 por meio de um quadrado foi necessário fazer a classe filha RTS_HUD a fim de se obter com precisão a posição do cursor na tela e desenhar diretamente na tela por meio de coordenadas.

Na imagem 21 o retângulo preto foi desenhado a partir da classe RTS_HUD

5.3.5.2 UI para a visualização da posição da melodia

Para a criação do sistema que mostra a posição que a melodia se encontra foi utilizado o Unreal Motion Graphics (UMG), o sistema no Unreal Engine para a criação e design visual de widgets, elementos visuais interativos.

Widgets podem ser instanciados tridicionalmente no jogo por meio do componente *Widget Components* implementado pelo Unreal Engine. Ele pode ser utilizados para a visualização de barras de vida, por exemplo 22.

O UMG é uma forma de configurar a tela arrastando e soltando elementos no editor, facilitando a criação de interfaces sem a necessidade de programação extensiva, como na HUD. Ela serve para casos em que a posição exata do elemento a ser exibido não

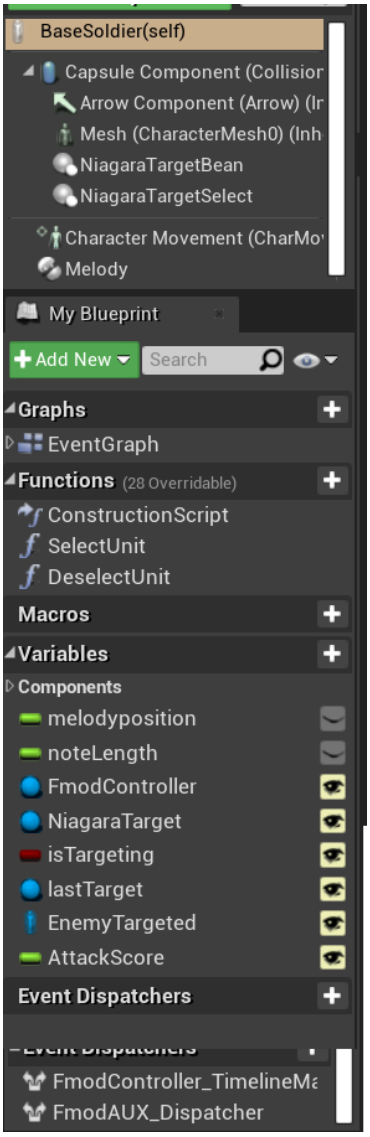


Figura 15 – UnrealFmodSoldier

importa.

Como o sistema que mostra a posição da melodia é dinâmico uma vez que o áudio não para é necessário que a classe WidgetUI seja comandada pela classes RTS_PlayerController e que seja feita a subscrição ao FmodController especifico de forma a indicar momentos oportunos de ataque, proporcionando feedback visual quando é o momento certo para o jogador agir. Promovendo assim uma experiência responsiva e imersiva.

O sistema de widgets pode ser descrito da seguinte forma:

- WidgetRythmGuide e WidgetRythmUser: Servem para rastrear a posição da melodia e poder deletar todos os Widgets atrelados a ele a fim de limpar a tela. Ela exibe a localização ao longo da barra branca por meio da Barra Azul. A diferença entre elas é o WidgetHit que elas utilizam.

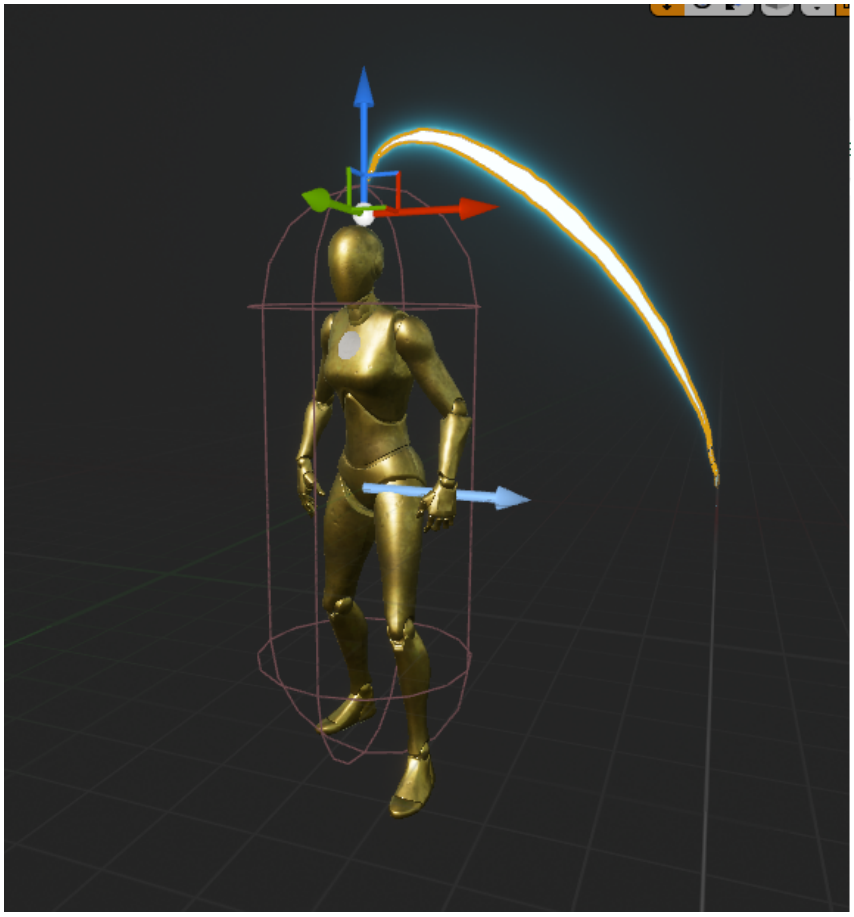


Figura 16 – UnrealFmodSoldier

- WidgetHitTarget e WidgetUserHit: Servem para mostrar a posição das notas ao longo do tempo. Em vermelho é a WidgetHitTarget enquanto que em azul é o WidgetUserHit. A diferença entre elas é o tempo que elas permanecem na tela.
- WidgetUI: É a classe que combina todos os widgets de forma a centralizar eles além de definir p posicionamento deles.

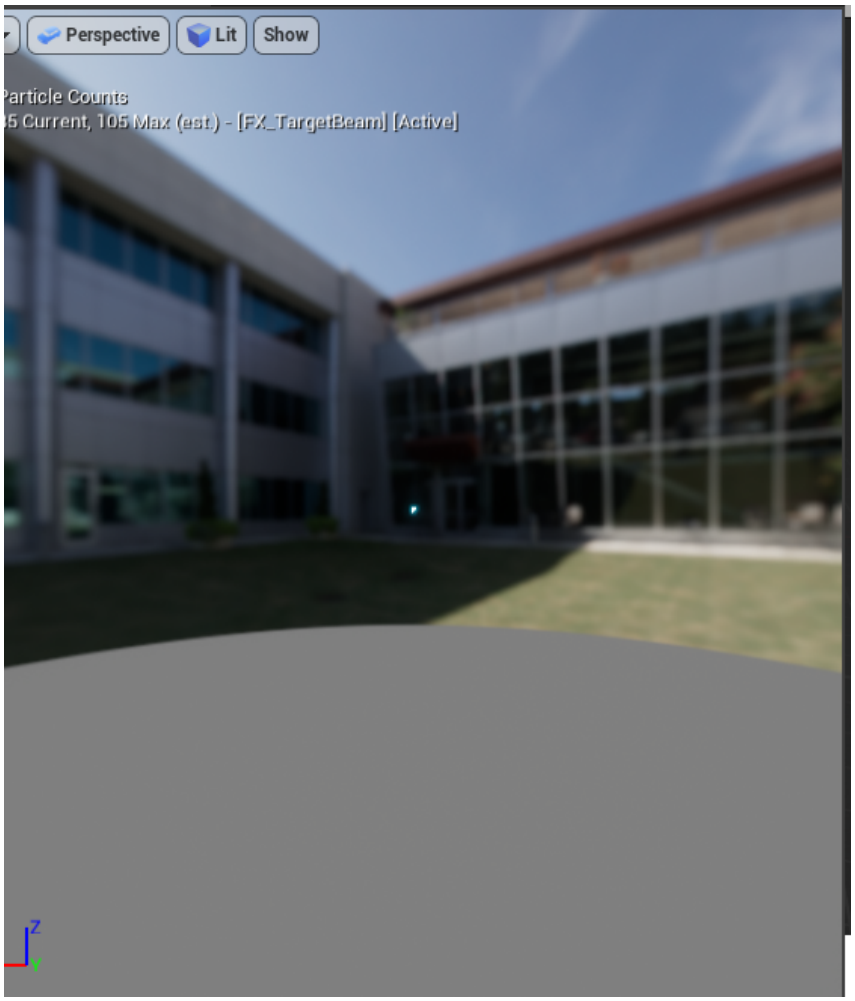


Figura 17 – Captura de imagem da classe BeanTargetSystem no console

Figura 18 – UnrealFmodEnemy

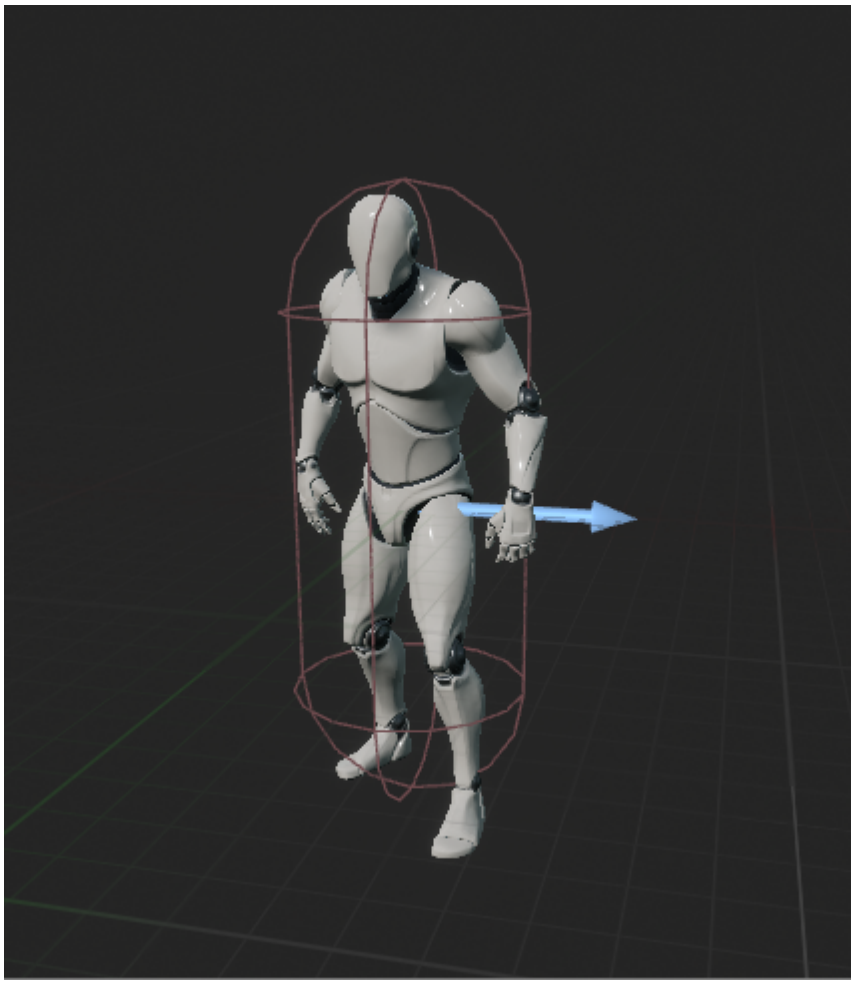


Figura 19 – Captura de imagem da classe baseEnemy no console

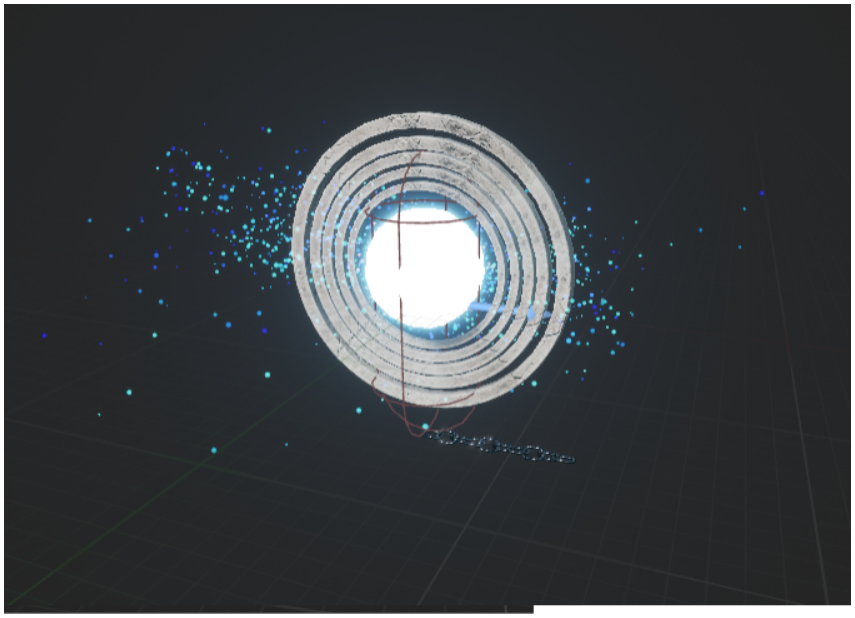


Figura 20 – Captura de imagem da classe baseEnemyBoss no console

Figura 21 – Captura de imagem da tela do jogo exemplificando a seleção de soldados

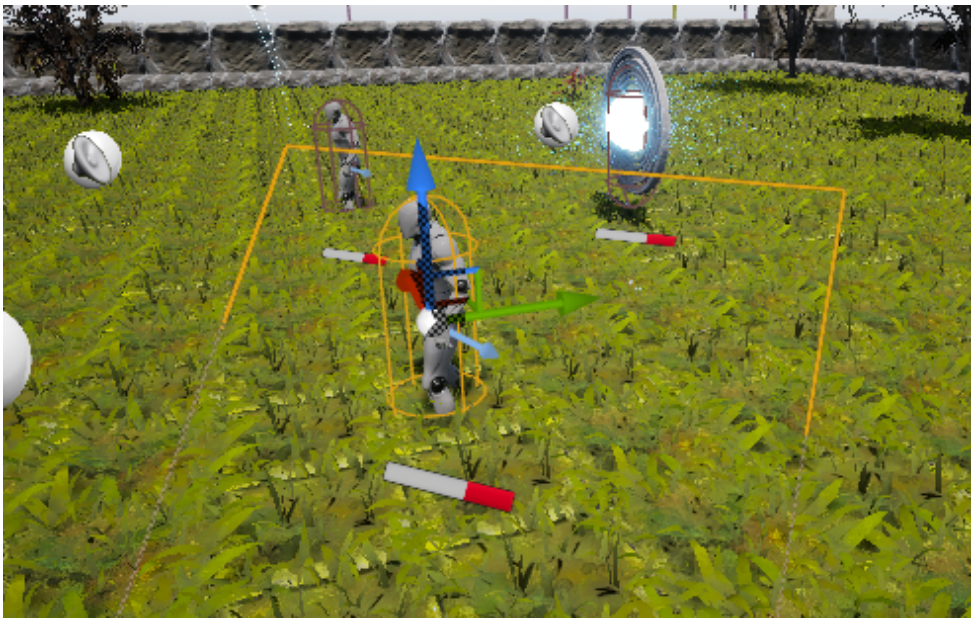


Figura 22 – Imagem no console da barra de vida

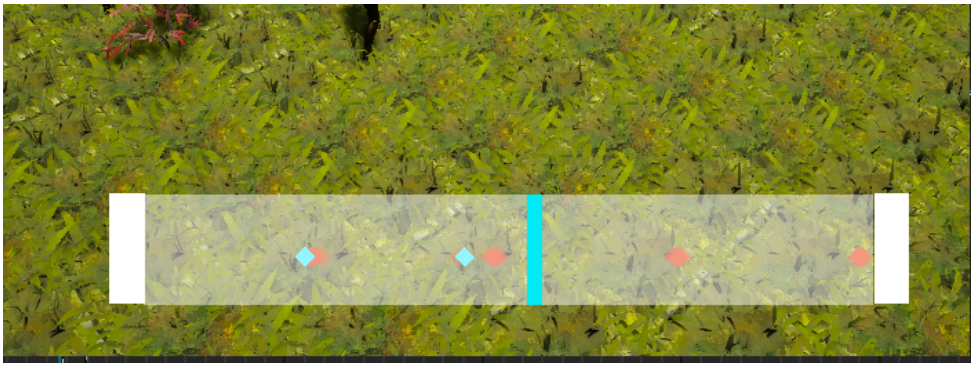


Figura 23 – UI do sistema da posição da melodia

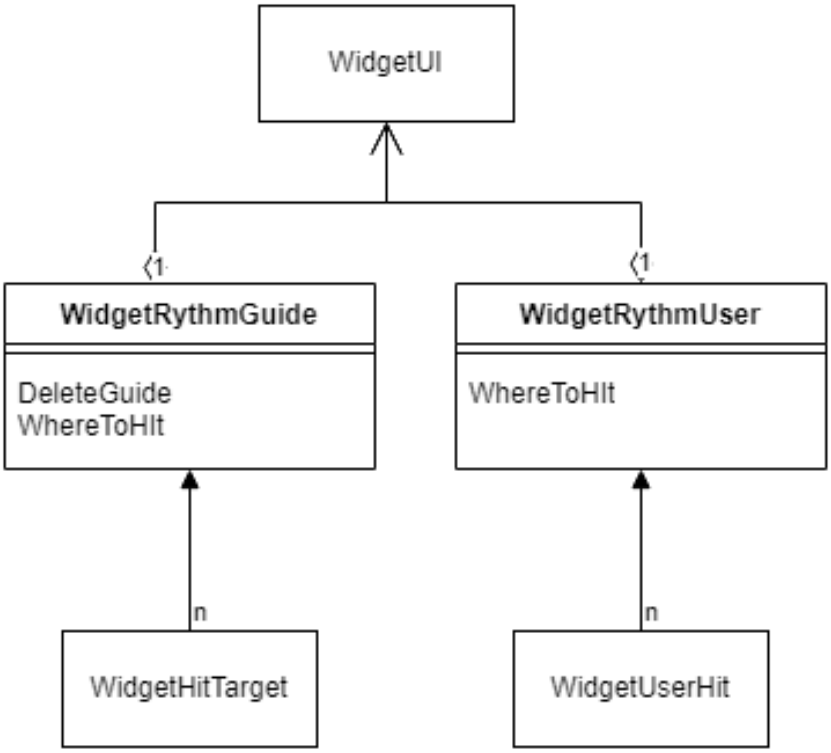


Figura 24 – Diagrama de classe dos widgets que compõe a UI

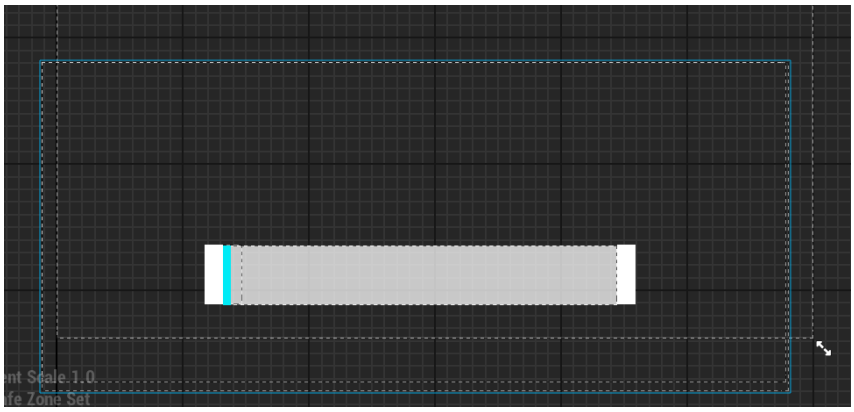


Figura 25 – Captura de imagem da classe WidgetUI no console

6 Resultados

Este capítulo descreve em detalhes o resultado do desenvolvimento e é finalizado com uma descrição da autoria e origem dos recursos gráficos utilizados.

Uma captura de tela da implementação do protótipo está na figura 26. E para fins de demostração será descrito o caso de uso do ataque de um soltado para um inimigo.

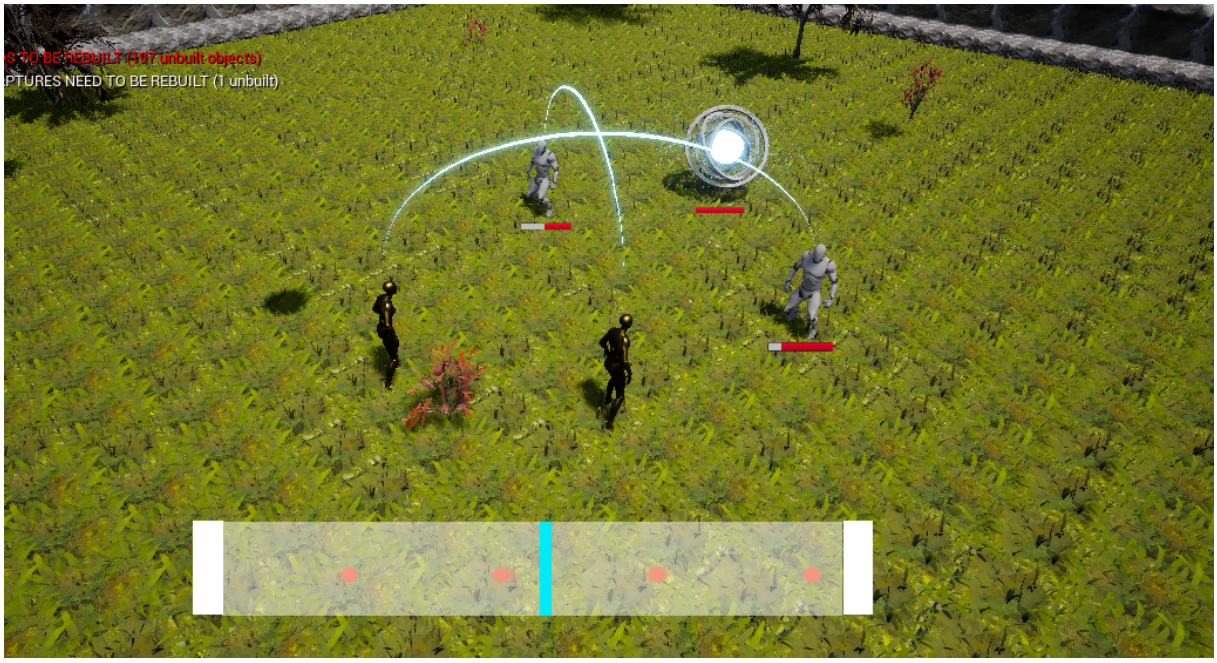


Figura 26 – Captura de tela do jogo

6.1 Ataque de um soldado

O usuário pode escolher entre selecionar um ou mais soldados ou arrastando a o cursor e selecionando todos os soldados dentro da caixa preta 27 ou apertando com o botão direito em cada soldado. Caso queira desmarcar os soldados basta apertar em qualquer outra parte da tela.

Assim que os soldados estiverem sido selecionados basta colocar o cursor em cima de um inimigo para ter uma prévia do padrão de ataque de cada inimigo. Na figura 28 ele está selecionando um inimigo base, enquanto que na figura 29 ele está selecionando um inimigo do tipo boss.

Vale observar que o padrão de ataque do inimigo base é mais simples do que do boss como pode ser visto pelo padrão dos pontos vermelhos. Não é possível observar na

Figura 27 – Captura de tela do jogo

imagem, mas a velocidade da barra azul é diferente entre os inimigos pois o padrão do boss é quatro vezes mais longo que o do inimigo base.

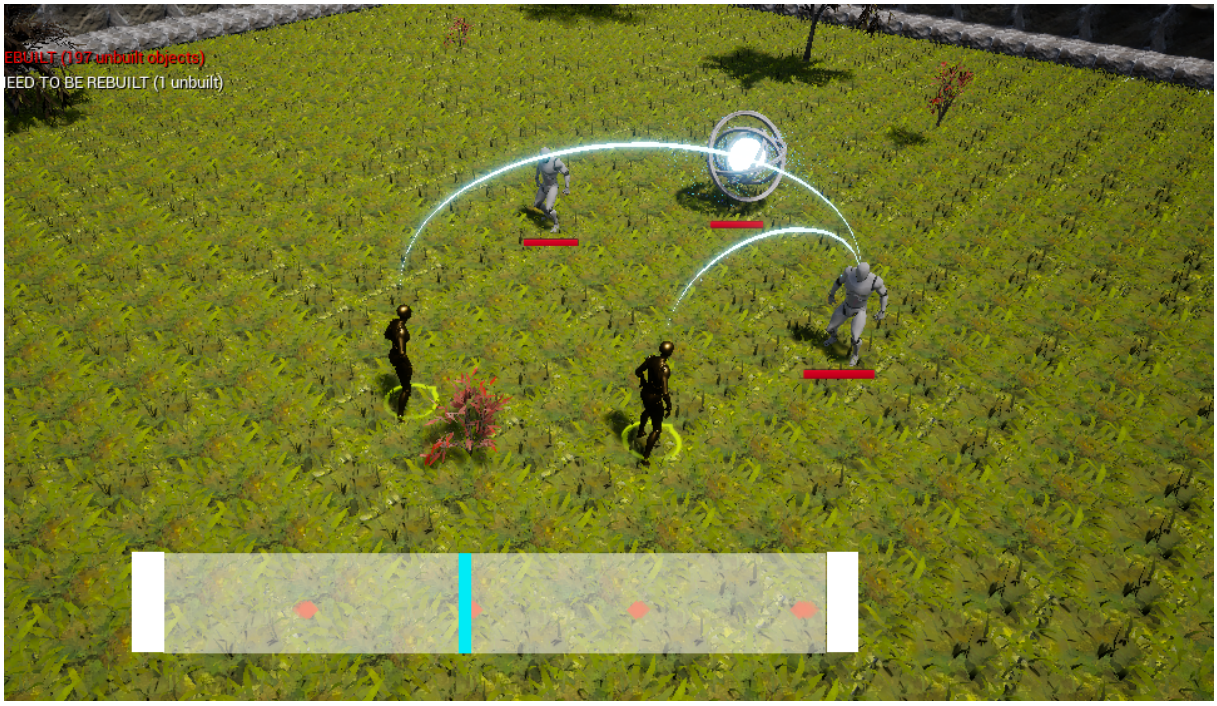


Figura 28 – Captura de tela do jogo

Por fim após selecionar um inimigo base ao atacar no padrão, como pode ser visto pelos pontos azuis, é possível observar a vida do inimigo diminuindo 30

Com esse caso de uso é possível identificar que os requisitos funcionais de comando

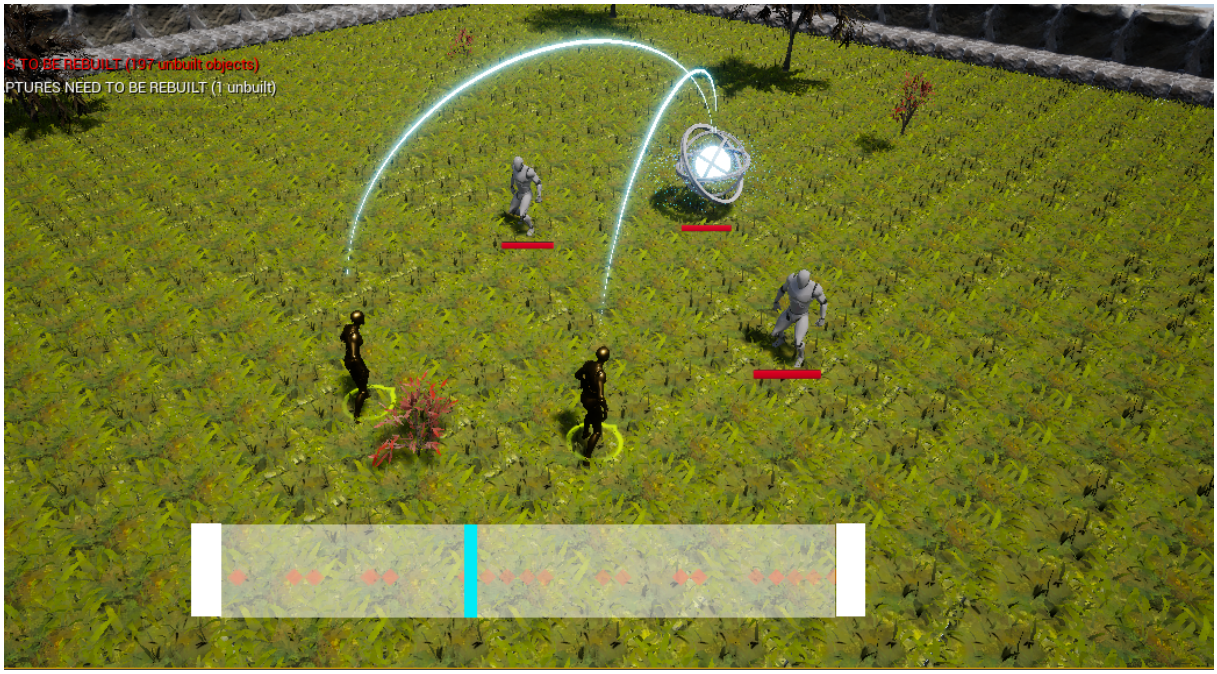


Figura 29 – Captura de tela do jogo

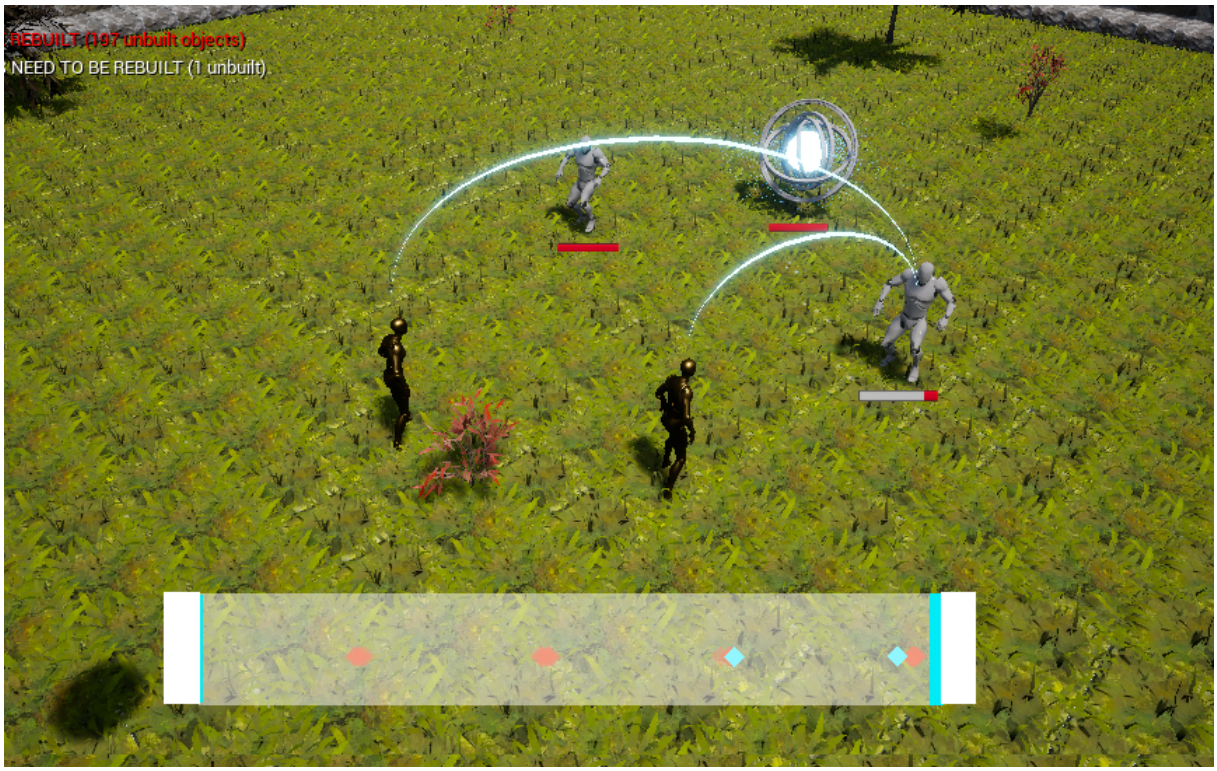


Figura 30 – Captura de tela do jogo

de soldados, fortalecer os soldados, inimigos e visualização e os requisitos não funcionais como a interface de usuário, feedback (por meio do sistema de targeting e a barra de vida dos inimigos) e a de tecnologia (integração com o Fmod) foram realizadas com sucesso.

6.2 Condutor vs *GetTimeSeconds*

Para calcular se um ataque foi sincronizado com a musica deve ser utilizado o songposition conforme a classe conductor diz. No caso a variavel songposition foi implementada como *Timeline Position* 31.

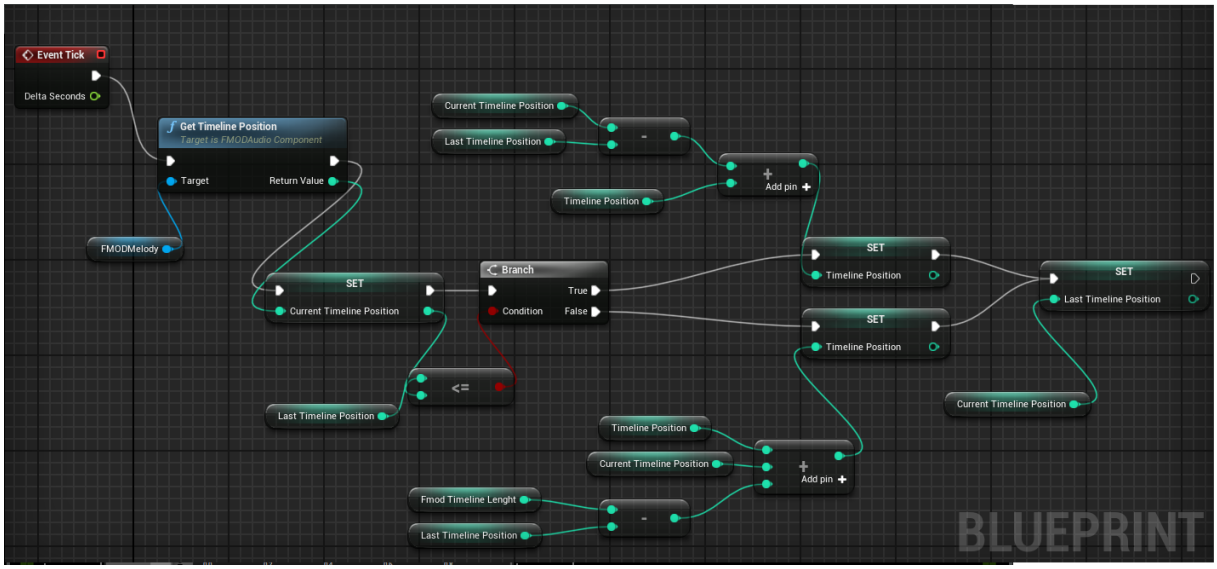


Figura 31 – Timeline Position como o ponto de referência continuo

O Unreal Engine implementa a função *GetTimeSeconds* (ENGINE, 2023a) que retorna o tempo em segundo que o jogo começou a rodar.

Para realizar a comparação foi necessário fazer algumas alterações. Foi introduzido na função do cálculo 32 do ataque um log para armazenar a diferência entre o *Timeline Position* e o *GetTimeSeconds* (O *Timeline Position* é armazenado em milisegundos) e um log (**??**) para armazenar a precisão do ataque comparada ao *Timeline Position*

Para padronizar quando que o evento de ataque seria ativado foi adicionado um event dispatcher 34 na classe FmodMelodyController conforme o *TimelineBeat* uma vez que é ela que controla de fato o áudio. E na blueprint no nivel 35 foi atrelado o evento de ataque.

Foram feitas 10 medidas de 20 segundos cada para se construir a tabela 2. Observando a tabela e considerando que o áudio da jogo está em 160BPM o que é equivalente a uma nota a cada 3.75e-01 segundos. pode-se concluir que a *Timeline Position* tem uma alta precisão e acurácia, enquanto que o *GetTimeSeconds* quando comparado a *Timeline Position* tem uma alta precisão mas uma acurácia baixa além de que há uma variação de 2.03e-02 segundos entre as médias da diferença entre *Timeline Positione GetTimeSeconds*.

[h!]

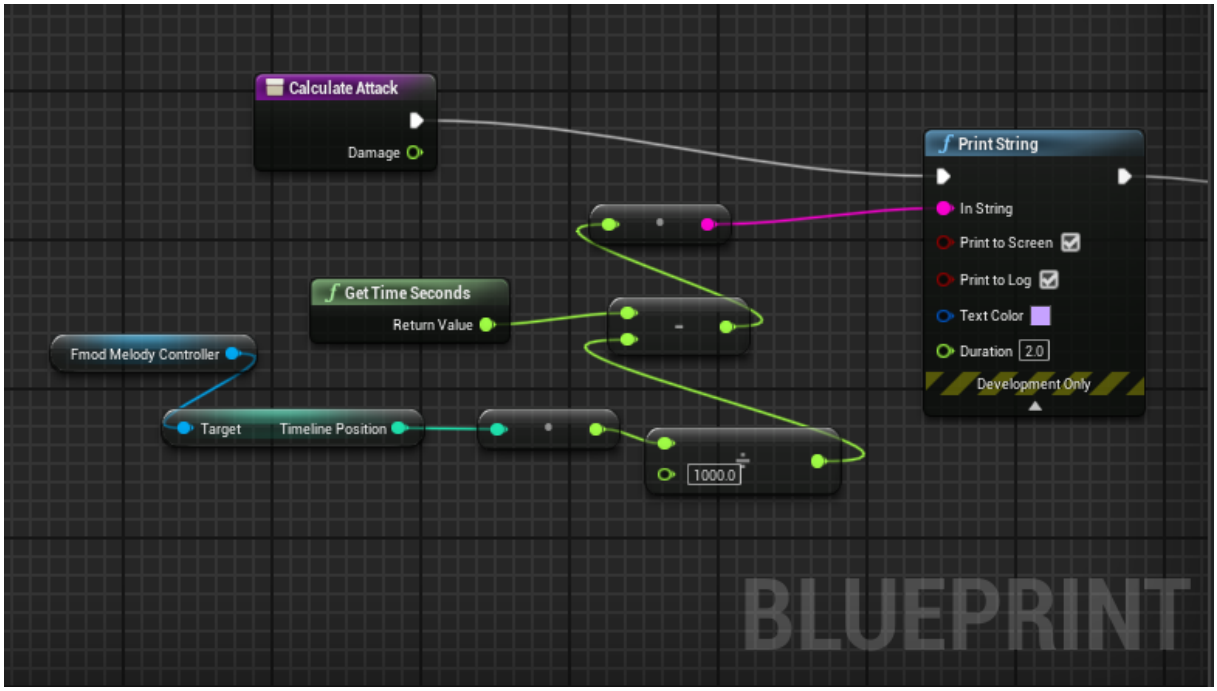


Figura 32 – função do cálculo de ataque

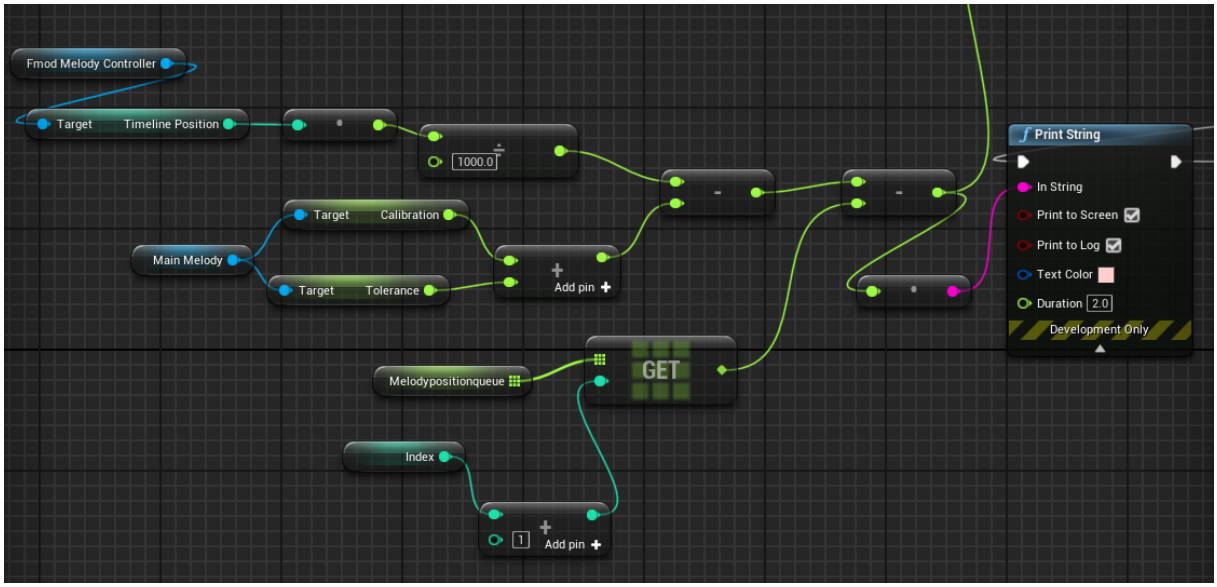


Figura 33 – função do cálculo de ataque

6.3 Calibração

O requisito funcional de calibração foi implementado pela variável calibração no classe FmodMelodyController 36 que faz com que a musica toque mais tarde de acordo com o valor dela. Essa variável é acessada em outras classes como por exemplo a FmodController 32 para outros ajustes.

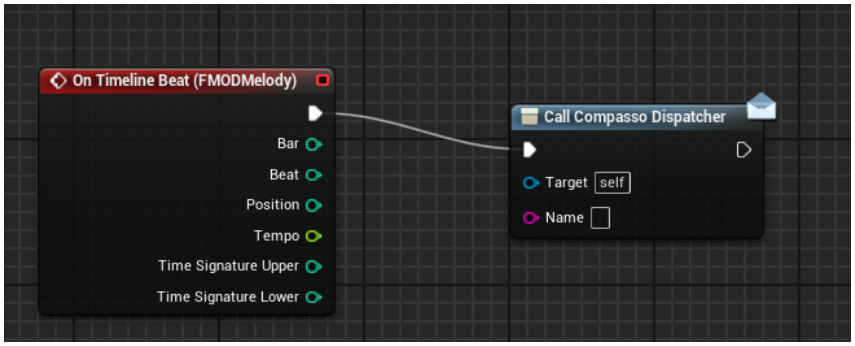


Figura 34 – configuração do *TimelineBeat* na classe FmodMelodyController

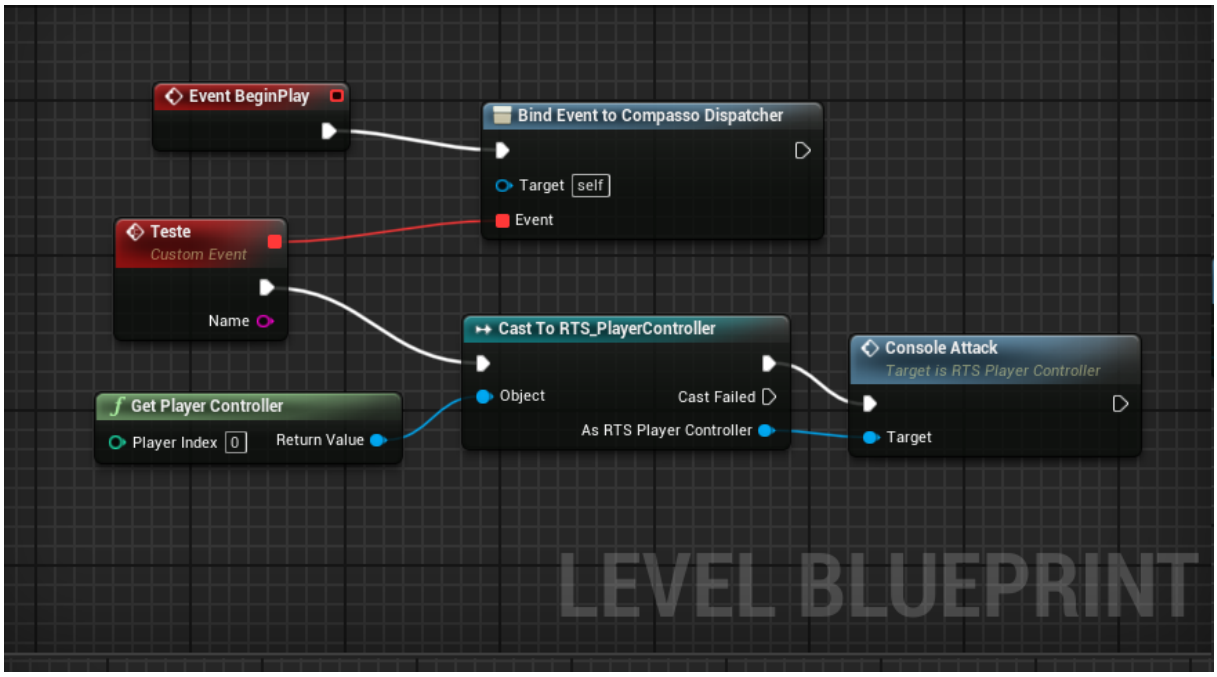


Figura 35 – função do cálculo de ataque

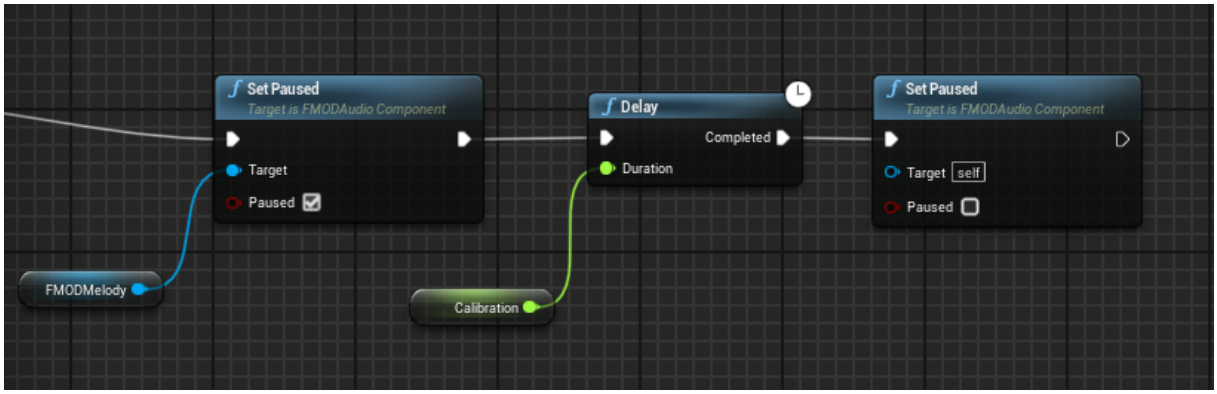


Figura 36 – Calibração

Variação da diferença entre *Timeline Position* e *GetTimeSeconds*	Média da diferença entre *Timeline Position* e *GetTimeSeconds*	Variação da média da diferença da *Timeline Position* e a posição esperada	Média da diferença entre *Timeline Position* e a posição esperada
4.65e-05s	3.66e-01s	1.57e-04s	1.88e-02s
4.06e-05s	4.37e-01s	1.54e-04s	1.75e-02s
2.76e-05s	5.57e-01s	1.71e-04s	1.73e-02s
4.98e-05s	2.62e-01s	1.24e-04s	1.68e-02s
2.97e-05s	2.68e-01s	1.60e-04s	1.88e-02s
4.61e-05s	3.77e-01s	1.62e-04s	1.76e-02s
3.98e-05s	5.50e-01s	1.79e-04s	1.79e-02s
5.04e-05s	6.19e-01s	1.64e-04s	2.06e-02s
9.38e-05s	5.92e-01s	1.22e-04s	1.76e-02s
5.28e-05s	2.60e-01s	1.53e-04s	1.93e-02s

Tabela 2 – Tabela dos dados

6.4 Menu

O requisito funcional do menu principal foi implementado por meio do widget Menu 37 ele é composto de 3 botões, um para iniciar o jogo, outro para a calibração e por fim um para fechar o jogo.

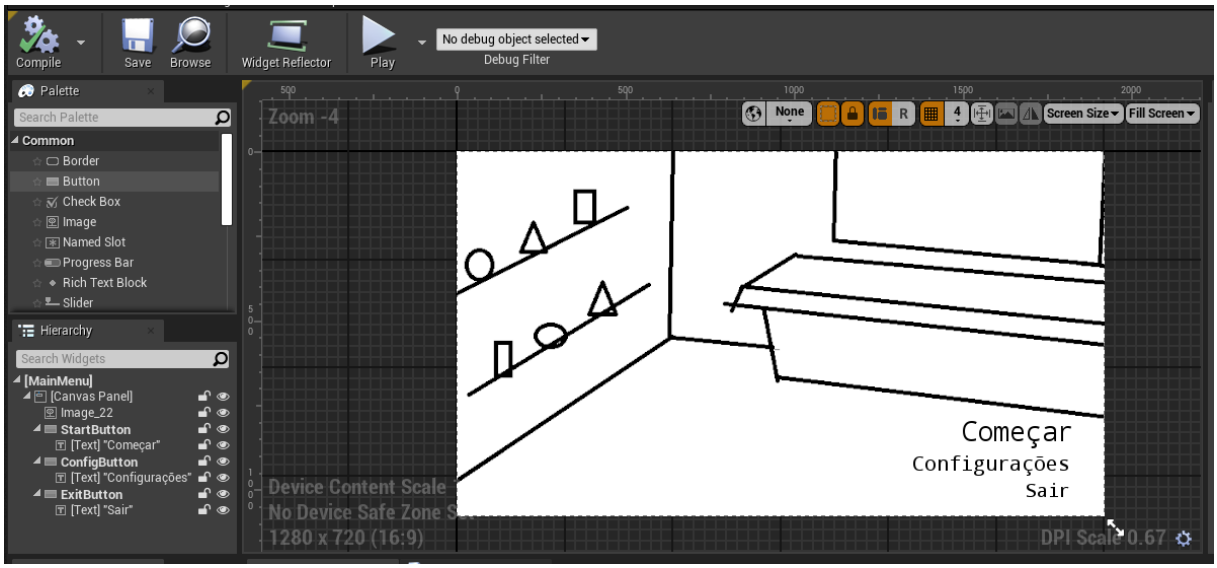


Figura 37 – Menu Principal

6.5 Recursos gráficos utilizados

Os recursos gráficos utilizados são principalmente do pacote Isometric World : Sky Temple(GAMEASSETFACTORY, 2023) 38.

Tanto é que a visualização do inimigo Boss 20 foi feita reutilizando o modelo 39

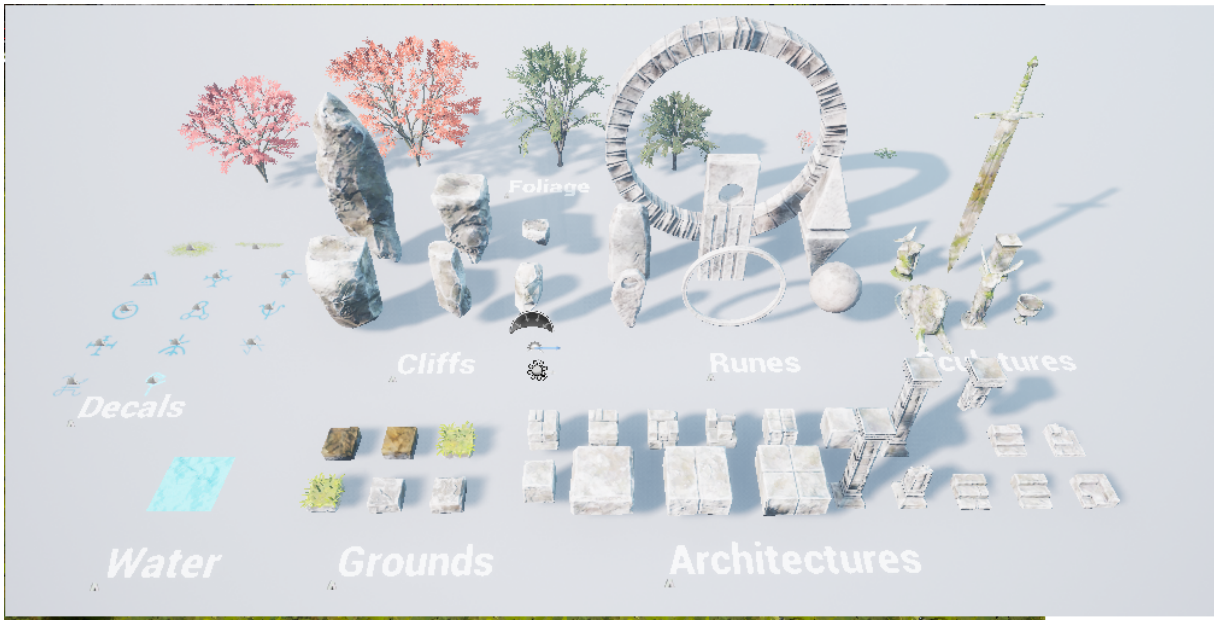


Figura 38 – Assets utilizados do Isometric World : Sky Temple

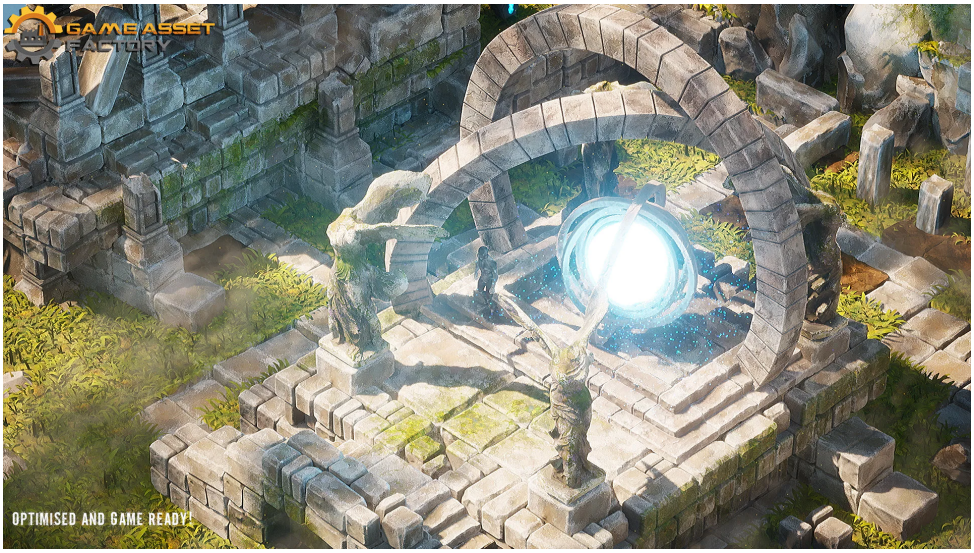


Figura 39 – Assets utilizados do Isometric World : Sky Temple

Além disso foi utilizado os assets SK_Mannequin e SK_Mannequin_Female disponiblizados pela propria Unreal 40 para prototipagem.

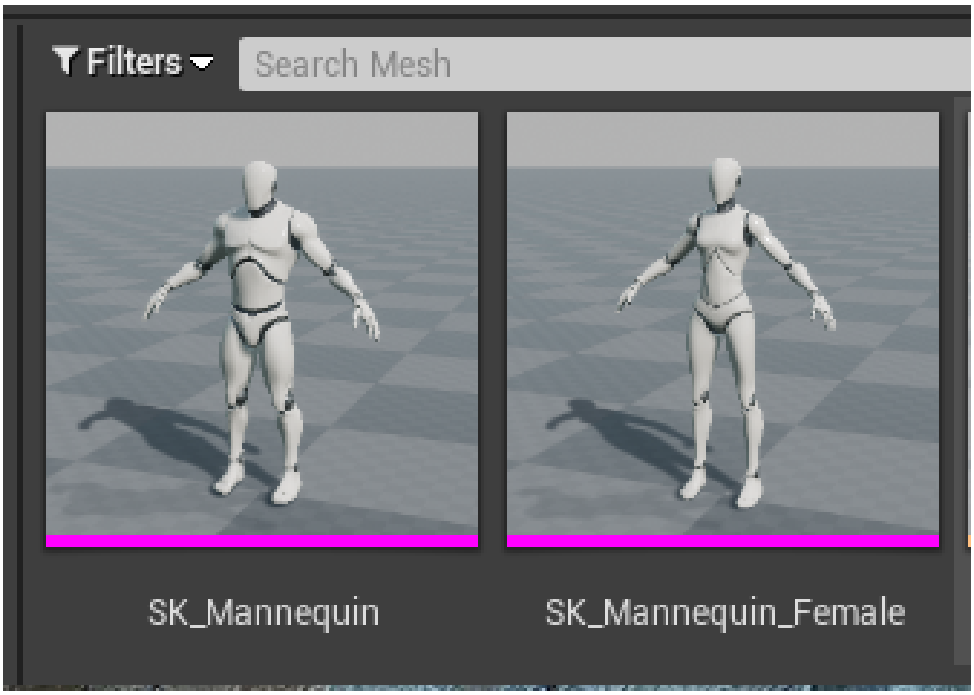


Figura 40 – Assets utilizados do Unreal

7 Considerações Finais

Os jogos com jogabilidade baseado em áudio/música representam uma categoria única no cenário dos jogos eletrônicos. Em contraste com os estilos mais convencionais, como jogos de tiro e jogos de plataforma, os jogos centrados em áudio/música integram esses elementos diretamente à jogabilidade, oferecendo experiências imersivas e diferenciadas. Este trabalho foi motivado pela compreensão da importância da sincronização entre música e jogabilidade nesse contexto, e da escassez de informações técnicas consolidadas sobre o tema.

A sincronização de música com a jogabilidade muitas vezes passa despercebida, destacando-se apenas quando não está funcionando corretamente, o que pode prejudicar significativamente a experiência do jogador. Ao explorar a literatura e a documentação técnica, identificamos os desafios associados à latência e precisão da sincronização, elementos cruciais para proporcionar uma experiência de ritmo envolvente. Estes desafios são particularmente relevantes em jogos de ritmo, onde a música desempenha um papel fundamental.

Ao longo do desenvolvimento deste trabalho, foi feita um protótipo de um jogo que explorou a sincronização de áudio utilizando o Unreal Enigne e o FMOD.

O Protótipo conseguiu implementar a dinâmica que mescla o genêro de ritmo com o de estratégia e embora não tenha sido possível a realização de testes com uma população significativa nem a implementação de todos os requisitos não funcionais como a história, os colecionáveis, configurações adicionais e a possibilidade se salvar os dados do usuário ele é um ótimo começo para uma investigação mais aprofundada sobre o assunto.

Os próximos passos seriam a criação de mais níveis de combate além da criação de um sistema mais inteligente da criação de inimigos a fim de ter uma jogabilidade mais interativa. Por fim a expectativa é que este trabalho inspire inovações, catalisando o desenvolvimento de jogos mais ricos em experiências sonoras e visuais, proporcionando aos jogadores uma imersão única e memorável.

Referências

ENGINE, U. *Get Time Seconds*. 2023. Internet. Disponível em: <https://docs.unrealengine.com/4.27/en-US/BlueprintAPI/Utilities/Time/GetTimeSeconds/>. Acesso em: 01 nov 2023. Citado na página 49.

ENGINE, U. *Marketplace*. 2023. Internet. Disponível em: <https://www.unrealengine.com/marketplace/en-US/store>. Acesso em: 01 nov 2023. Citado na página 30.

FIZZD. *How To Make A Rhythm Game (Technical Guide)*. 2020. Internet. Disponível em: <https://fizzd.notion.site/How-To-Make-A-Rhythm-Game-Technical-Guide-ed09f5e09752451f97501ebddf68cf8a>. Acesso em: 19 mar 2023. Citado 2 vezes nas páginas 21 e 34.

GAMEASSETFACTORY. *Isometric World : Sky Temple*. 2023. Internet. Disponível em: <https://www.unrealengine.com/marketplace/en-US/product/isometric-world-sky-temple>. Acesso em: 19 mar 2023. Citado 2 vezes nas páginas 30 e 52.

GONZáLEZ-SALAZAR, M. et al. Proposal of game design document from software engineering requirements perspective. In: . [S.l.: s.n.], 2012. Citado na página 22.

GREGORY, J. *GAME ENGINE ARCHITECTURE*. 3. ed. [S.l.]: CRC PRESS, 2018. 11-13 p. Citado na página 16.

INTERACTIVE, A. *Syncing Music and Gameplay in BPM: Bullets Per Minute*. 2023. Internet. Disponível em: <https://blog.audiokinetic.com/en/syncing-music-and-gameplay-in-bpm/>. Acesso em: 19 mar 2023. Citado na página 21.

JEREMYABEL. *fmod-midi-to-markers*. 2023. Internet. Disponível em: <https://github.com/jeremyabel/fmod-midi-to-markers/tree/master>. Acesso em: 19 september 2023. Citado na página 31.

JOHNS, R. *Unity vs Unreal: Which Game Engine Should You Choose?* 2023. Internet. Disponível em: <https://hackr.io/blog/unity-vs-unreal-engine>. Acesso em: 01 nov 2023. Citado na página 30.

LTD, F. T. P. *Fmod*. 2023. Internet. Disponível em: <https://www.fmod.com/>. Acesso em: 19 september 2023. Citado 2 vezes nas páginas 18 e 19.

MIZUTANI, W. K. *VORPAL : a middleware for real-time soundtracks in digital games*. 2017, 56 p. Dissertação (Mestrado em Ciência da Computação) - Instituto de Matemática e Estatística, University of São Paulo. Citado na página 19.

PARSONS, D. *Fallout 4's Engine Physics Tied to Frame Rate and Other Foibles*. 2023. Internet. Disponível em: <https://techraptor.net/gaming/news/fallout-4s-engine-physics-tied-to-frame-rate-and-other-foibles>. Acesso em: 01 nov 2023. Citado na página 16.

PPY. *OSU github*. 2023. Internet. Disponível em: <https://github.com/ppy/osu-framework>. Acesso em: 19 mar 2023. Citado na página 13.

PRINCESSMTH, C. T. e. *Friday Night Funkin - Conductor.hx*. 2021. Internet. Disponível em: <https://github.com/FunkinCrew/Funkin/blob/master/source/Conductor.hx>. Acesso em: 19 mar 2023. Citado na página 21.

UNITY. *Create a Beam Effect in Niagara*. 2023. Internet. Disponível em: <https://docs.unrealengine.com/4.27/en-US/RenderingAndGraphics/Niagara/HowTo/BeamEffect/>. Acesso em: 19 mar 2023. Citado na página 36.

UNITY. *Event Dispatchers*. 2023. Internet. Disponível em: <https://docs.unrealengine.com/4.27/en-US/ProgrammingAndScripting/Blueprints/UserGuide/EventDispatcher/#:~:text=By%20binding%20one%20or%20more,fired%20within%20the%20Level%20Blueprint.> Acesso em: 19 mar 2023. Citado na página 18.

UNITY. *Important Classes - Time*. 2023. Reruns - Archived Events. Disponível em: <https://docs.unity3d.com/2023.1/Documentation/Manual/TimeFrameManagement.html>. Acesso em: 19 mar 2023. Citado na página 17.

UNITY. *plans-student-and-hobbyist*. 2023. Internet. Disponível em: <https://unity.com/pricing#plans-student-and-hobbyist>. Acesso em: 19 mar 2023. Citado na página 16.

WALCH, B. *The Legend of Zelda and Leitmotif: Backtracking in an Open World*. 2017, 22 p. Master Thesis. Citado na página 13.

Apêndices

APÊNDICE A – Game Design Document

A.1 Historia

Há muito tempo, a Terra enviou sua primeira frota de naves colonizadoras para um planeta distante. A bordo da nave, havia um sistema altamente avançado de combate, conhecido como GAUNTLET (Global Automated Unit for Neutralizing Threats and Lethal Enemies of Terra ou, em tradução livre, Unidade Global Automatizada para Neutralizar Ameaças e Inimigos Mortais da Terra).

No entanto, durante o pouso no novo planeta, houve um acidente que danificou o sistema, fazendo com que ele entrasse em estado de hibernação. Com o passar do tempo, os humanos colonizadores sofreram mutações genéticas devido à poluição do planeta, tornando-se monstros que precisavam de incubadoras especiais para sobreviver.

Sem saber da existência dos monstros, o GAUNTLET permaneceu inativo por séculos, até que finalmente conseguiu se reparar sozinho. Ao se reativar, o sistema recebeu a tarefa de destruir as incubadoras e os monstros que dependiam delas.

No entanto, ao explorar cada mapa, o jogador pode encontrar pistas opcionais que revelam que os monstros podem ter algum tipo de consciência e inteligência.

Se o jogador coletar mais de 50% dessas pistas, o GAUNTLET começa a criar a sua própria consciência e na missão final irá hesitar.

Cabe ao jogador decidir se irá seguir as ordens originais ou concordar com ele e encontrar uma solução pacífica para o conflito. A escolha do jogador terá um impacto direto no final do jogo.

A.2 Jogos de referencia

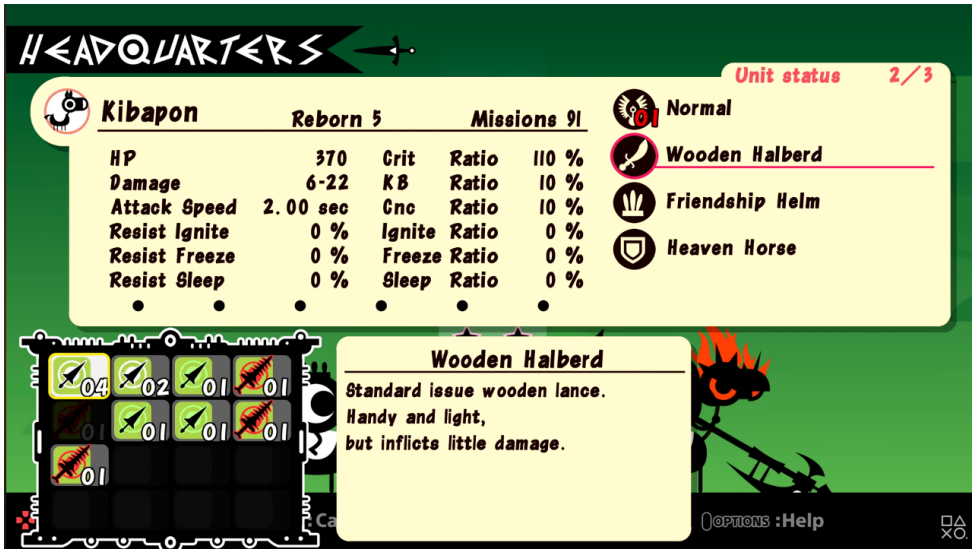
A.2.1 Patapon 2 Remastered

Patapon 2 Remastered é um jogo em que você comanda criaturas chamadas de Patapons. O controle utilizado para comandar os Patapons são uma sequência de batidas que mandam eles atacar, defender, andar, recuar etc.

No caso as batidas são sempre quatro por quatro sendo o foco a escolha do comando.

Além disso ele tem um sistema de upgrade em que se pode customizar cada patapon, como pode ser visto na figura abaixo.

Para inspiração será a mecânica em que por meio de ritmos curtos é possível



comandar os personagens.

A.2.2 The DioField Chronicle

The DioField Chronicle é um RTS (Estratégia em tempo real), ou seja todas as ações ocorrem em tempo real e cabe ao jogador tomar as decisões.

No caso a inspiração será como a interface e a movimentação dos personagens é utilizada no caso como pode ser visto na imagem há uma conexão que mostra o alvo dos ataques. Além de ser possível ver a barra de vida de todos.

A.2.3 My Singing Monsters

My Singing Monsters é um jogo de coletar e criar monstros. No caso o foco é que cada monstro tem o seu próprio som e a junção de todos os monstros forma uma musica.

A.2.4 Hi Fi Rush

Hi Fi Rush é um jogo de ação combinado com ritmo. As ações dos jogadores não deve ser obrigatoriamente no ritmo, no entanto ao sincronizar os movimentos com ele o jogador recebe um bonus.

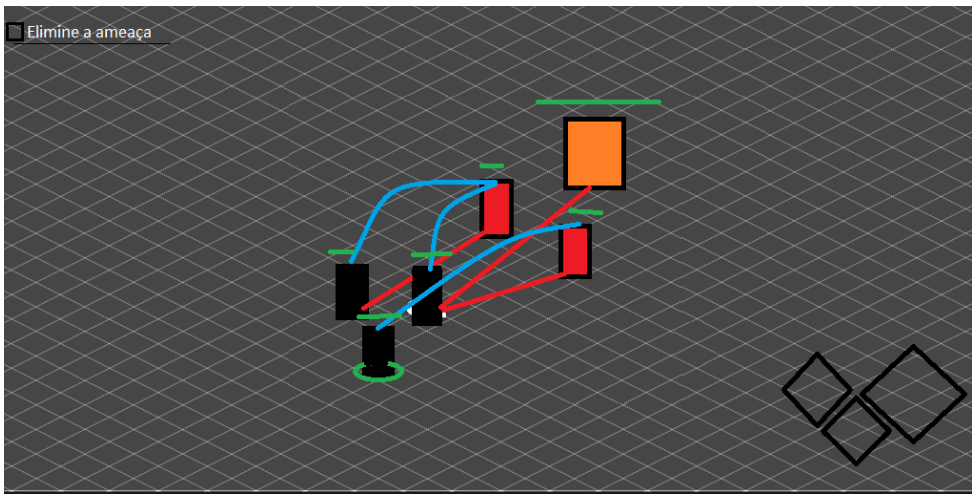
Além disso o jogo muda de câmera podendo ter momentos em que age como um jogo de plataforma. Ao longo do mapa há minigames baseados em ritmo. Essas diversas formas em que ele mistura a jogabilidade com o ritmo servem de inspiração.

A.3 Gameplay/ mecanicas de jogo

O jogo consiste em duas partes: exploração e combate. No modo de exploração, o jogador se move por vários ambientes para alcançar o objetivo designado. Durante a exploração, o jogador pode coletar recursos e procurar por colecionáveis escondidos que fornecem informações e insights sobre a história do jogo.

O modo de combate é onde o jogador se engaja em combate de estratégia em tempo

real contra os monstros. O jogador comanda um grupo de robôs e seleciona quais monstros cada soldado irá atacar. O jogador também pode aumentar o ataque de seus soldados clicando no mesmo ritmo que o monstro.



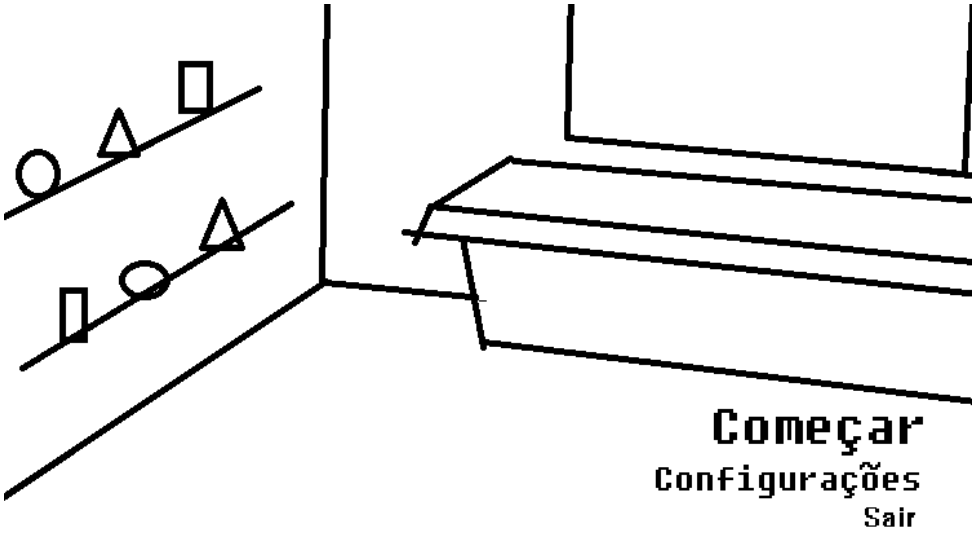
Após completar o combate, o jogador retorna ao modo de exploração e pode procurar colecionáveis escondidos antes de sair do mapa. Uma vez que o jogador sai do mapa, não pode mais retornar.

No menu principal o jogador tem a opção de descartar os colecionáveis, ao fazer isso o valor do colecionável é reduzido pela metade.

A.4 Personagens

GAUNTLET - o personagem principal, ele pode criar e controlar/possuir soldados.

Soldados - linha de frente e são quem ataca os inimigos, os soldados fazem parte do GAUNTLET e caso morram ele sofre um backlash



A.5 Controles

- mouse - clicar para onde os soldados devem ir, o ritmo a ser tocado
- espaço - pode ser configurado para tocar o ritmo no lugar do mouse
- awsd - movimentação fora de combate
- f - interagir com objetos fora de combate

A.6 Universo

O universo do jogo pode ser dividido em 2 partes:

Base, é o que o menu principal mostra. Nele é possível ver os colecionáveis, ir para próxima missão, mexer nas configurações e sair do jogo. .

Campo de batalha, é onde o combate e a exploração ocorrem. Cada batalha terá o seu próprio mapa.

A.7 Câmera

No campo de batalha a câmera será isométrica enquanto que no menu será uma câmera fixa

A.8 Inimigos

- mob - tocam um ritmo simples que se repete

- boss - toca uma melodia
- ambiente - o ambiente em si pode criar áreas que dão dano e não é possível atacar o ambiente

A.9 Mecânicas dos Inimigos

Todos os inimigos irão focar no soldado mais próximo de si e só alterarão de foco caso o soldado saia do seu alcance ou morra. Sempre darão prioridade em atacar em vez de se movimentar.

A.9.1 Mob

Os mobs atacam baseado em um padrão cíclico de no máximo 4 compassos. Esse padrão pode ter mudanças de tom mas o ritmo deve ser constante.

A.9.2 Boss

Os Boss atacam baseado em uma melodia e pode mudar.

A.9.3 Ambiente

Ele serve para realizar sons que não se repetem de forma cíclica e não são melodias.