JURASSIC JUMP – LIGA Fase 2

1. Visão Geral do Projeto

Este projeto é um jogo de plataforma 2D desenvolvido para o processo seletivo da LIGA Facens. O jogador controla um dinossauro que deve saltar entre plataformas e chegar ao final do nível, com o tempo de conclusão sendo registrado junto do número de pontos coletados para criar um elemento de competição.

2. Ferramentas Utilizadas

- Unity 2022.3.4: Desenvolvimento do jogo, gráficos e lógica.
- Git: Controle de versão.

3. Build e Plataforma

O jogo foi gerado como um APK para Android (ARMv7) e testado em dispositivos móveis, com funcionalidade esperada.

4. Principais Classes

Player: Gerencia o controle do personagem (movimentação, pulo, input, renderização e colisão) através de outras classes como PlayerMove, PlayerInput, PlayerRender, PlayerJump e PlayerDetection.

Função principal: Coordenar os comportamentos do personagem durante o jogo.

GameManager: Gerencia o ciclo de vida do jogo, incluindo vidas do jogador, tempo de jogo, pontos coletados e transição entre cenas.

Função principal: Controle de estado do jogo e navegação entre menus e cenas.

HudManager: Controla a interface do usuário, exibindo informações como tempo, pontos e vidas, além de menus de jogo e resultados.

Função principal: Atualizar HUD e menus.

PlayerUI: Responsável pela interação entre os botões da interface e o comportamento do jogo.

Função principal: Gerenciar eventos de interface, como iniciar o jogo ou voltar ao menu.

DamagePlayer e Portal:

1. DamagePlayer: Detecta colisões que causam dano ao jogador, reduzindo vidas.
2. Portal: Detecta a vitória do jogador ao alcançar o final do nível.

5. Tempo Gasto

O desenvolvimento do projeto levou aproximadamente 14 horas, distribuídas entre estudos, design, codificação e testes.

6. Maiores Dificuldades

Integração com Unity Ads: O sistema de monetização e análise com dashboard exigiu ajustes para funcionar corretamente na versão inicial.

7. Interação de Coleta e Dano

Nesta seção, vamos detalhar as implementações que permitem ao jogador interagir com objetos coletáveis e entidades que causam dano, como gemas, moedas e armadilhas.

7.1. Interface IDamageable

A interface IDamageable é usada para objetos que podem causar dano ao jogador. A ideia é que qualquer objeto que implemente essa interface tenha um comportamento específico para aplicar dano ao jogador.

7.2. Interface ICollectible

A interface ICollectible é implementada em objetos que podem ser coletados pelo jogador, como gemas e moedas. Qualquer objeto que implemente essa interface deve fornecer uma lógica para quando for coletado.

7.3. Coleta de Gemas e Moedas

Os itens coletáveis no jogo são representados pelas classes GemCollect e CoinCollect. Ambas implementam a interface ICollectible, permitindo que o jogador interaja com esses objetos.

- GemCollect: Coleta de gemas que adiciona 5 pontos ao jogador.
- CoinCollect: Coleta de moedas que adiciona 1 ponto ao jogador.

7.4. Detecção de Jogador

O script PlayerDetection é responsável por detectar a colisão do jogador com objetos coletáveis e de dano. Ele usa as interfaces ICollectible e IDamageable para garantir que, quando o jogador entra em contato com um objeto que implementa uma dessas interfaces, a ação correspondente é executada.

7.5. Dano ao Jogador

A classe DamagePlayer implementa a interface IDamageable. Ela interage com o GameManager para reduzir a vida do jogador quando o dano é aplicado.

8. Unity Analytics e Unity Ads

Nesta seção, abordaremos o uso do Unity Analytics e Unity Ads implementados no projeto para monitorar o comportamento dos jogadores e, opcionalmente, monetizar o jogo com anúncios.

8.1. Unity Analytics

O Unity Analytics foi configurado para coletar dados sobre as interações dos jogadores com o jogo. Ele permite o acompanhamento de métricas como tempo gasto no jogo, número de vezes que o jogador completa um nível, e possíveis desistências. Isso possibilita futuras melhorias com base em dados reais do comportamento dos jogadores.

Principais funcionalidades:

- Inicialização do Unity Services no método Start.
- Coleta dados de gameplay core do Analytics como **gameStart** e **gameEnd** através do método AnalyticsService.Instance.**StartDataCollection**().

Esses dados podem ser consultados no painel do Unity Analytics, oferecendo uma visão detalhada sobre o comportamento dos jogadores e possibilitando a tomada de decisões fundamentadas para futuras atualizações do jogo.

8.2. Unity Ads

O Unity Ads foi integrado como uma opção de monetização para o projeto. Embora ainda em estágio inicial, ele permite a exibição de anúncios intersticiais no jogo, que podem ser ativados em pontos estratégicos para gerar receita.

Principais funcionalidades:

- Inicialização do Unity Ads com o androidGameId.
- Carregamento e exibição de anúncios intersticiais.
- Implementação de listeners para gerenciar o estado dos anúncios (carregamento, exibição, falhas e cliques).

9. Possíveis Melhorias Futuras

Embora o desenvolvimento do **Jurassic Jump** não vá continuar após o término deste processo seletivo, algumas melhorias podem ser consideradas para futuras versões ou projetos similares, caso se decida expandir o jogo.

9.1. Desacoplamento de Classes

Atualmente, algumas classes do jogo ainda dependem diretamente do **GameManager**, o que pode aumentar o acoplamento e dificultar a manutenção. Uma possível melhoria seria implementar um sistema de **listeners** no **GameManager**, permitindo que outras classes interajam com ele de maneira desacoplada. Isso facilitaria futuras alterações e melhorias, além de tornar o código mais modular e escalável.

9.2. Correção de Bugs de Performance

Existem alguns problemas de performance que poderiam ser solucionados em futuras atualizações:

- **HUD:** A interface do usuário às vezes aparece antes que a próxima tela seja totalmente carregada. Uma correção para esse comportamento garantiria que a HUD só fosse exibida no momento correto.

- **Joystick:** O joystick apresenta, ocasionalmente, um bug em que o input já está configurado para a direita ao iniciar o jogo, mesmo sem interação do jogador. Embora esse código tenha sido importado, uma revisão poderia corrigir esse problema para evitar comportamento inesperado.

9.3. Expansão com Novas Fases e Conteúdo

Futuras versões do jogo poderiam incluir:

- **Novas Fases:** Com o objetivo de proporcionar mais desafio e variedade, fases mais longas e complexas poderiam ser adicionadas.

- **Novos Coletáveis:** Power-ups e debuffs, aproveitando as interfaces já implementadas (**ICollectible**), poderiam aumentar a profundidade das mecânicas de jogo.
- **Mais Objetos de Dano:** Adicionar mais elementos que causem dano ao jogador, utilizando a interface **IDamageable**, poderia aumentar a dificuldade e complexidade das interações.
- **Inimigos com Interface de Ataque:** O desenvolvimento de inimigos que implementam uma interface **IEnemy** permitiria a adição de diferentes tipos de dano, podendo ser reutilizada entre diversos inimigos com comportamentos variados.

9.4. Refinamento na Coleta de Dados com Unity Analytics

A coleta de dados no jogo poderia ser aprimorada com o **Unity Analytics**, para fornecer informações mais detalhadas sobre o comportamento dos jogadores e experiências de **crash.**

9.5. Melhorias no Sistema de Anúncios

O sistema de anúncios poderia ser aprimorado com a adição de anúncios **rewarded**, onde o jogador é recompensado por assistir aos anúncios. Além disso, ajustes poderiam ser feitos para exibir anúncios em momentos mais oportunos, como após a conclusão de fases ou durante pausas, de forma a melhorar a experiência do jogador sem prejudicar o fluxo do jogo.

10. Acesso ao Código-Fonte

O código-fonte do projeto **Jurassic Jump** está disponível para visualização e download em um repositório no GitHub. Neste repositório, você encontrará todas as classes, interfaces e scripts utilizados no desenvolvimento do jogo

Link para o repositório: GitHub - Phase Two Repository