INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA DE SÃO PAULO

TIFANY LUIZA DE JESUS MOREIRA

THE LEGACY OF THE THREE RUNES: UM GAME BROWSER DESENVOLVIDO COM PHASER 3 E JAVASCRIPT

CAMPOS DO JORDÃO

2025

RESUMO

O projeto *The Legacy of the Three Runes* consiste em um jogo de plataforma 2D com temática de fantasia sombria. O jogador assume o papel da "Tempestade", o único capaz de confrontar o Fire Wizard que utilizou o poder proibido do Legado das Três Runas. A narrativa se baseia na libertação das Almas Infernais, presas pela magia sombria do Mago. O objetivo primário em cada fase (Runa) é coletar todas as almas perdidas, cuja luz confere a força necessária para quebrar as Runas e, finalmente, enfrentar o Fire Wizard.

O jogo foi desenvolvido utilizando a *framework* **Phaser 3**, com arquitetura baseada em cenas e componentes (entidades), garantindo modularidade e escalabilidade.

Palavras-Chave: Phaser 3, Jogo de Plataforma 2D, Modularidade, Escalabilidade, Jogo de Plataforma.

SUMÁRIO

1 INTRODUÇÃO

Este relatório tem como finalidade documentar o desenvolvimento do jogo digital *The Legacy of the Three Runes*, um projeto concebido como um *platformer* 2D que explora elementos de ação e fantasia sombria. O jogo narra a jornada da protagonista, a "Tempestade", que busca restaurar o equilíbrio em um mundo devastado pelo poder proibido das Três Runas, manipulado por um Fire Wizard. A estrutura do projeto é baseada em uma progressão clara através de três estágios (Runas), onde a coleta de almas serve como mecanismo central para o avanço da narrativa, resultando em uma batalha de chefe final que testa todas as mecânicas de ataque utilizadas durante as três fases. A seguir, serão detalhados os objetivos, a justificativa para sua criação, os aspectos metodológicos e o aporte teórico que fundamentaram a execução deste trabalho.

1.1 Objetivos

O principal objetivo deste projeto é desenvolver e implementar um jogo de ação-plataforma 2D completo, que sirva como evidência da proficiência na utilização de *frameworks* de desenvolvimento de jogos. Especificamente, busca-se:

1. **Implementar um sistema de física 2D funcional** capaz de gerenciar colisões, gravidade e movimentação fluida do jogador (incluindo pulo, caminhada e corrida).
2. **Desenvolver uma arquitetura de código modular** baseada em Programação Orientada a Objetos (POO), encapsulando a lógica de entidades (Player, Knight, Boss) e a transição de estado de cena (Scenes).
3. **Criar e integrar um sistema de combate básico**, incluindo detecção de *hitbox* por *frame* de animação e gestão de vida (HP) e dano para o jogador e inimigos.
4. **Implementar sistemas de Inteligência Artificial (IA) simples** para os inimigos, abrangendo padrões de patrulha e sequências de ataque baseadas em distância, garantindo um desafio progressivo.

1.2 Justificativa

A criação de *The Legacy of the Three Runes* justifica-se pela necessidade de aplicar e consolidar conhecimentos de desenvolvimento de software em um ambiente de projeto prático e criativo. O domínio de uma *framework* como a Phaser 3 demonstra a capacidade de integrar lógica de programação, *design* de interação e gestão de *assets* multimídia (gráficos e áudio) para produzir uma experiência interativa coesa. Além disso, o projeto serve como um estudo de caso na implementação de IA rudimentar e na modelagem de um sistema de progressão de jogo que equilibra a exploração (coleta de almas) com o combate, refletindo as melhores práticas do *level design* em jogos de plataforma.

1.3 Aspectos Metodológicos

A metodologia de desenvolvimento adotada seguiu os princípios da **Programação Orientada a Objetos (POO)**, sendo cada personagem (jogador, inimigo, boss) representado por classes que herdam de objetos do Phaser e contêm seus próprios estados e comportamentos. O projeto adere ao padrão de **Arquitetura de Componentes** inerente ao Phaser, separando a lógica de jogo em Scenes para modularizar os níveis e telas (Menu, Introdução, Fases e Boss). O aporte teórico se baseia em:

1. **Mecânicas de Jogos de Plataforma Clássicos:** O sistema de controle de movimento e física foi ajustado para oferecer *feedback* tátil imediato, priorizando a precisão em plataformas e saltos.
2. **Narrativa:** Utiliza o formato de *cutscene* de introdução (com *typewriter* e *assets* de áudio) para estabelecer o tom sombrio e a motivação do protagonista, seguindo a estrutura narrativa de "jornada do herói".
3. **Design de Batalha de Chefe (Boss Fight Design):** O Fire Wizard (Boss) implementa uma máquina de estados simples, onde os padrões de ataque são determinados pela distância do Player (Tempestade).

2 METODOLOGIA

2.1 Considerações Iniciais e Ferramentas

O projeto foi desenvolvido como um jogo de plataforma 2D utilizando a *framework* Phaser 3. As dimensões padrão do jogo foram definidas em 1000 pixels de largura por 700 pixels de altura, garantindo uma proporção consistente para os elementos de *pixel art*. O estilo visual predominante de escolha é o *dark fantasy pixel art*, com inspiração na série de Games de Castlevania.

2.2 Ferramentas Utilizadas:

- Phaser 3: *Framework* voltada para o desenvolvimento de jogos 2D.
- Vite: Ferramenta de *building*.
- JavaScript: Linguagem de programação principal utilizada, estruturada em módulos para melhor organização e importação de classes.
- HTML/CSS: Para a estrutura base do contêiner do jogo e estilização da fonte medieval (*MedievalSharp*).

2.3 Arquitetura do Projeto

A arquitetura do projeto é baseada no padrão de Cenas (Scenes) do Phaser, controladas pelo arquivo principal src/main.js. As funcionalidades do jogo são encapsuladas em classes de entidades (Objetos de Jogo) e classes de Cenas (Níveis/Telas). Essa separação facilita a transição de estado e o gerenciamento de recursos, como música e UI, de forma isolada para cada etapa do jogo.

Path do arquivo	Descrição
src/main.js	Inicializa o Phaser, define as configurações globais (1000x700) e lista a ordem de todas as cenas.
src/scenes/	Contém a lógica de cada tela/nível (MenuScene.js, GameScene.js, FinalBossScene.js, etc.), gerenciando a criação do cenário, UI, inimigos e regras de colisão
src/entities/	Contém as classes que definem o comportamento dos personagens (Player.js, Knight.js, Boss.js) por meio de métodos como update(), takeDamage(), e patrol()
src/scenes/PreloadScene.js	Carrega todos os *assets* (sprites, áudio, fundos) e registra todas as animações (ex: anim_idle, knight_walk) do jogo, garantindo que estejam prontas para uso, sem necessidade de repetição de declaração e construção das animações nas classes utilizantes.
public/assets/	Armazena todos os *assets* de mídia do jogo (imagens, spritesheets e sons).

2.4 Descrição e Fluxo do Projeto

O jogo segue um fluxo linear e progressivo, em que:

1. **PreloadScene**: Inicia o jogo, carregando todos os recursos visuais e auditivos e configurando as animações globais do Phaser e configurando, também, as animações dos personagens dos jogos (os spritesheets).

2. **MenuScene**: Tela inicial, ponto de partida e acesso à tela de Controles que exibe um modal com todos os controles e teclas correspondentes do game.
3. **IntroductionScene**: Sequência cinemática (Cutscene) que estabelece a premissa narrativa através de um monólogo com efeito *typewriter*, animando a entrada do personagem e introduzindo, de forma breve, a trama vingativa da protagonista (Storm) referente ao antagonista (Wizard).
4. **GameScene (Runa 1)**: Primeira fase de plataformas e combate, onde o jogador deve coletar a primeira porção de Almas (Fires) para avançar, enfrentando os Cavalheiros (Knights). O jogador deve coletar as 6 almas para passar de runa, sendo a morte dos enemies apenas um fator dificultador para conseguir coletar as almas com sucesso sem reiniciar a fase.
5. **SecondRuneScene (Runa 2)**: Nível de plataforma com design vertical e inimigos dispostos de forma a exigir mais precisão de pulo e combate. Este nível é mais característico por possuir uma maior dificuldade pela disposição estratégica do spawn dos enemies, visto que há um enemy por plataforma, dificultando um pouco a mobilidade do player para atacar os Knights e, posteriormente, coletar as Almas.
6. **ThirdRuneScene (Runa 3)**: Última fase antes do Boss, focada no combate contra grupos de Cavalheiros, testando a resiliência e a habilidade de ataque do jogador. O nível em questão, quando comparado aos outros dois, é o de caráter mais fácil e de menos complexidade na disposição dos Cavalheiros e Almas.
7. **FinalBossCutScene**: Cutscene final de diálogo, elevando a tensão antes do confronto direto com o antagonista.
8. **FinalBossScene**: Batalha final do game, introduzindo padrões de ataque um pouco mais complexos por parte do Fire Wizard, sendo estes baseados em uma IA básica que se estrutura a partir de uma máquina de estados que se baseia como ponto de partida a distância do Fire Wizard em relação ao player (Storm/Tempestade), com uma barra de vida dedicada ao vilão e tela de vitória após a derrota do Fire Wizard.
9. **PauseScene**: Uma cena sobreposta que permite pausar o jogo e oferece opções de Continuar, Reiniciar Fase, Reiniciar Jornada (que irá direcionar o jogador para a primeira runa, GameScene.js) e Ir para o Menu.

2.5 Tempestade (Storm) - Descrição de Atributos

Atributo	Detalhe	Origem
HP Inicial	6 (3 corações completos, onde cada coração equivale a 2 de HP)	Player.js
Velocidade Padrão	200 (Caminhada)	Player.js
Velocidade Corrida	350 (ativada com a tecla [Shift])	Player.js
Força de Pulo	-520 (velocidade vertical instantânea)	Player.js
Ataque Rápido	[Espaço], Dano: 1, Frames de Hit: 3 a 8. Atinge múltiplos inimigos por golpe.	Player.js
Ataque Forte	[E], Dano: 2, Frames de Hit: 1 a 4. Possui maior alcance (attackRange: 140) e é mais lento.	Player.js
Mecânica de Dano	Recebe *knockback* e invulnerabilidade temporária (efeito *blink*) por 1 segundo ao ser atingido.	Player.js

2.6 Knight (Enemy) - Descrição de Atributos

Atributo	Detalhe	Origem
HP	3	Knight.js
Dano	1 (no jogador)	Knight.js
Comportamento	Patrulha linear limitada pela patrolRange. Distância de Patrulha varia entre 70 (Runa 1), 30 (Runa 2) e 50 (Runa 3) pixels do ponto inicial. Possui um tempo de espera (waitTime: 1000ms) ao inverter a direção.	Knight.js
Ataque	Inicia a animação de ataque se o jogador estiver a menos de 100px. O acerto é checado a partir do frame 2 da animação, com *cooldown* de 2000ms.	Knight.js

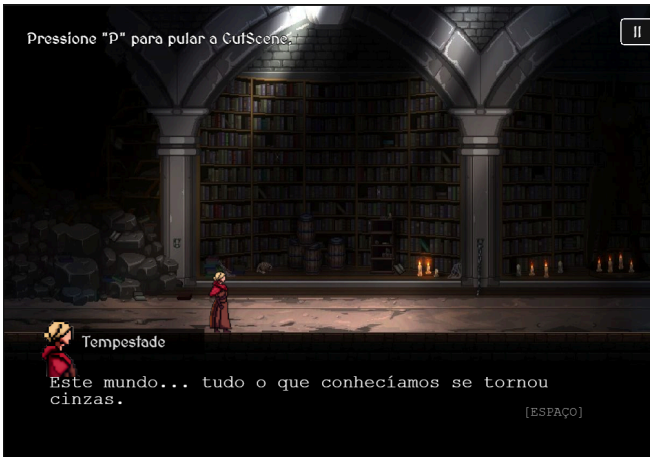
2.7 Boss: Fire Wizard - Descrição de Atributos

Atributo	Detalhe	Origem
HP	30	Boss.js
Dano	1 (por acerto)	FinalBossScene.js
Padrão de Combate	Máquina de estados baseada na distância do jogador, em que:	Boss.js
	1. Perto (< 200px): 60% Jato de Fogo; 40% Pulo para Trás.	Boss.js
	2. Médio (200px a 550px): 20% Investida; 10% Bola de Fogo; 70% Idle/Patrulha.	Boss.js
	3. Longe (> 550px): 50% Bola de Fogo; 50% Perseguição.	Boss.js
Ataques Especiais	Bola de Fogo: Lança um projétil teleguiado de velocidade alta. Investida: Carga rápida em direção ao jogador. Jato de Fogo: Ataque de curto alcance e alto DPS se o jogador estiver muito próximo.	Boss.js, FinalBossScene.js

3 RESULTADOS OBTIDOS

Abaixo seguem algumas demonstrações de capturas de tela do jogo e alguns exemplos de animações (spritesheets) dos personagens do jogo.

Tela de Menu (MenuScene) - Captura de tela



Cena de introdução (IntroductionScene) - Captura de tela

Primera Runa (GameScene) - Captura de tela

Pause (PauseScene) - Captura de tela

Animação de dano da Tempestade (Storm) - Pixabay

Animação das Almas (fogos) coletáveis - Autoral

3.1 Funções Essenciais Implementadas

Função	Arquivo	Descrição
updateHealthUI(currentHealth)	GameScene.js, SecondRuneScene.js, FinalBossScene.js	Gerencia a representação visual da vida no HUD. Cada imagem de coração é "cortada" (setCrop()) para mostrar vida cheia (2 HP), meia vida (1 HP) ou vazia (0 HP).
checkPlayerAttacks()	GameScene.js, Player.js	Loop de detecção de acerto do jogador em entidades inimigas. Verifica se o jogador está atacando (isAttacking), se o frame de animação é ativo (isFrameActive()) e se o inimigo está dentro do alcance e na direção correta.
patrol()	Knight.js	Patrulhamento autônomo do inimigo Cavalheiro. Calcula se o inimigo excedeu o patrolRange ou se encontrou um bloqueio, acionando um tempo de espera antes de inverter a direção.
attackFireball(player)	Boss.js	Sequência de ataque mágico: o Boss faz a animação de lançamento e, após um *delay*, um projétil é instanciado e lançado com velocidade na direção atual do jogador (velocityFromRotation).

startNextLevelTransition()	GameScene.js, SecondRuneScene.js	Controla a passagem de fase após a coleta de todos os coletáveis (almas). Realiza fadeOut e transfere as variáveis de estado (score, health) para o novo cenário.
handlePlayerDeath()	GameScene.js, SecondRuneScene.js, ThirdRuneScene.js	Responde ao HP zerado do jogador, parando sons, acionando o fadeOut da câmera e apresentando a tela de morte com a opção de reiniciar.

4 CONCLUSÃO

O projeto The Legacy of the Three Runes demonstrou a capacidade de desenvolver um jogo de plataforma 2D funcional e tematicamente coeso com a proposta usando a framework Phaser 3. A aplicação da Programação Orientada a Objetos para as entidades (Player, Knight, Boss) resultou em um código modular e de fácil manutenção e leitura, permitindo a separação clara entre lógica de jogo e a estrutura de cena. O sistema de progressão por Runas e a manutenção do estado do jogador (HP e Score) entre as cenas foram implementados com sucesso, proporcionando uma experiência contínua. A implementação da IA básica do Boss, baseada em máquina de estados e distância, adicionou complexidade e desafio à batalha final e uma robustez maior à proposta de entrega do jogo.

4.1 Sugestões para Possíveis Melhorias

1. **Sistema de Habilidades e Mana:** Introduzir habilidades mágicas secundárias para a Tempestade (além dos ataques básicos), como um ataque de projétil de relâmpago, consumindo um medidor de mana.

2. **Variedade de Inimigos:** Criar uma classe de inimigo à distância para as fases das Runas, complementando o Cavalheiro de combate corpo a corpo e forçando o jogador a adotar estratégias de aproximação variadas.
3. **Otimização da IA do Knight:** Melhorar a detecção de borda do Cavalheiro, implementando um raio de verificação de chão à frente para evitar quedas desnecessárias, permitindo patrulhas mais complexas em plataformas não uniformes.
4. **Sistema de Save/Load de Progresso:** Implementar um sistema de salvamento persistente usando o localStorage para que o jogador possa retomar o progresso a partir da última Runa desbloqueada após fechar a aplicação.
5. **Aprimoramento do HUD do Boss:** Adicionar um texto ou indicador visual na FinalBossScene durante a fase de "carga" do Boss para comunicar melhor o iminente próximo ataque.
6. **Reunir todos os componentes de UI e HUD em uma única classe:** Visto que os componentes de contador de Almas e exibição de vidas atuais são repetitivos ao longo das classes, seria ideal criar uma única classe para esse tipo de situação, em que no meu construtor haveria um parâmetro no qual eu pudesse diferencial de qual scene viria e, com esta informação, exibir o HUD de acordo com a fase atual. Por exemplo, se for na fase do boss final, exibir, além da vida do player, a vida do boss também.

REFERÊNCIAS

- **PHASER.** *Phaser 3 Documentation.* Disponível em: https://photonstorm.github.io/phaser3-docs/.
- **PIXABAY.** https://pixabay.com/sound-effects/search/flash%20attack
- **PISKEL.** https://www.piskelapp.com