Curso: Engenharia de Software

Professor: Adam Smith
Data de Início: 25/11/2024
Data de Conclusão: 02/12/2024
Integrantes da Equipe:

Matheus Ferreira Souza: Pesquisa, Implementação e Soluções/Desenvolvimento (Front e Back-end)

Lucas Aves Santana: Pesquisa, Implementação e Soluções/Desenvolvimento (Front e Back-end)

Kayron Gabriel Gomes Nascimento: Pesquisa, Documentação, Design, Implementação de Soluções/Desenvolvimento (Front)

Erick Ferreira Dos Santos: Pesquisa e Documentação

Marcos Vinicius Nunes Moreira: Pesquisa e Documentação

Documentação do Projeto: Pokémon Purple

Objetivos

O projeto "Pokémon Purple" teve como principal objetivo desenvolver um jogo RPG de combate que simulasse batalhas no universo Pokémon. O jogo deveria ser simples, divertido e memorável, atendendo às expectativas e exigências de aprendizado da disciplina. Inicialmente, havia dúvidas sobre qual tipo de jogo seria representado. Após discussões no grupo, decidiu-se por um sistema de batalha Pokémon devido à familiaridade e simplicidade do conceito, além de ser algo divertido e acessível para os usuários.

O desenvolvimento envolveu a criação de uma estrutura lógica sólida, implementando regras de batalha e eventos visuais que trouxessem vida ao jogo. O foco foi demonstrar habilidades tanto em lógica de programação quanto em design visual e interatividade.

Além disso, o objetivo foi demonstrar a importância do estudo contínuo, indo além do conteúdo apresentado em sala de aula, incentivando outros alunos a explorarem novas possibilidades e a não se limitarem apenas ao material proposto.

Recursos Existentes

Durante o desenvolvimento do projeto, os seguintes recursos foram utilizados:

1. **Ferramentas de Desenvolvimento:**
    - Compilador **Visual Studio Code** com extensões específicas:
    - C/C++
    - CMake Tools
    - [FF] Flutter Files
    - C/C++ Compile Run
    - C/C++ Extension Pack
    - C/C++ Runner
    - C/C++ Themes
    - Docker
    - PowerShell
    - Biblioteca **Raylib**: Escolhida devido à sua sintaxe clara e acessível, que facilitou a implementação dos elementos gráficos e de interação.
2. **Recursos Visuais:**
    - Imagens em formato PNG obtidas de diversos sites.
    - GIFs animados criados no **CapCut**.
    - Design gráfico desenvolvido no **Adobe Photoshop Pro**.
    - **Soluções Auxiliares:**
    - **ChatGPT**: Ferramenta de suporte para resolver problemas técnicos e criar soluções otimizadas.
    - **YouTube**: Fonte de tutoriais e referências, tanto para elementos visuais quanto sonoros.
3. **Recursos de Áudio:**
    - Músicas de fundo e efeitos sonoros extraídos do YouTube.

Guias de Estilo de Linguagem

O código foi estruturado seguindo os princípios da biblioteca **Raylib**, com ênfase em:

Modularidade: Cada funcionalidade principal foi implementada como uma função específica, facilitando a manutenção e a leitura do código.

Legibilidade: O uso de nomes descritivos para variáveis e funções garantiu que o código fosse fácil de entender.

Funções do código explicadas:

1. **DrawCenteredText**
   Exibe um texto centralizado horizontalmente em relação à largura da tela, em uma posição vertical especificada, com uma cor e tamanho de fonte definidos.
2. **DrawCenteredTexture**
   Desenha uma textura centralizada na tela, ajustando sua escala e posicionamento baseado nos valores de largura e altura da tela.
3. **DrawScaledTexture**
   Desenha uma textura ajustada à escala proporcional para caber na tela, considerando a largura e altura disponíveis.
4. **UpdatePlayerAction**
   Atualiza a ação atual de um jogador para uma ação específica (parado, ataque ou atingido), configurando a duração do estado da ação em quadros.
5. **HandlePlayerActions**
   Gerencia as ações de ataque e resposta dos jogadores. Detecta teclas pressionadas para executar ataques, reproduz sons, atualiza estados dos jogadores e calcula os danos aplicados. Além disso, reseta as ações dos jogadores ao estado parado após o término do contador de frames.
6. **DrawAttackEffect**
   Desenha um efeito visual de ataque com partículas circulares coloridas em posições aleatórias ao redor de um ponto central.
7. **FlashPokemonPlayer1**
   Faz o Pokémon do jogador 1 piscar alternando entre visível e invisível por um número de frames definido.
8. **FlashPokemonPlayer2**
   Faz o Pokémon do jogador 2 piscar alternando entre visível e invisível por um número de frames definido.
9. **AtaqueJogador1**
   Inicia o efeito de piscagem para o Pokémon do jogador 2, indicando que foi atingido.

10. **AtaqueJogador2**
    Inicia o efeito de piscagem para o Pokémon do jogador 1, indicando que foi atingido.

11. **main**
    Função principal que configura o jogo, inicializando a janela, os dispositivos de áudio, e carregando os recursos como músicas, texturas, e sons. Ela também define os estados iniciais dos jogadores e gerencia o loop principal do jogo, incluindo a lógica de transição entre telas e atualizações do estado do jogo.

Funções de inicialização, carregamento e música:

12. **InitWindow**
    Cria a janela do jogo com o título especificado e define a largura e altura da tela.

13. **InitAudioDevice**
    Inicializa o dispositivo de áudio, permitindo o uso de sons e músicas no jogo.

14. **LoadSound**
    Carrega um arquivo de som (como MP3 ou WAV) para ser usado em eventos como ataques ou defesas.

15. **LoadMusicStream**
    Carrega um arquivo de música em streaming, permitindo a reprodução contínua e controlável (como música de fundo).

16. **PlayMusicStream**
    Começa a reproduzir uma música carregada no formato de streaming.

17. **SetMusicVolume**
    Define o volume da música em um valor proporcional (de 0.0 a 1.0).

18. **LoadImageAnim**
    Carrega uma animação no formato de imagem, retornando o número total de quadros disponíveis.

19. **LoadTextureFromImage**
    Converte uma imagem carregada em uma textura utilizável no jogo.

20. **LoadTexture**
    Carrega uma textura diretamente de um arquivo de imagem.

Variáveis globais e estruturas:

21. **Scenario**
    Estrutura que define um cenário do jogo, incluindo um nome e a textura do plano de fundo correspondente.

22. **Pokeball**
    Estrutura que define uma Pokébola, contendo um nome, uma cor associada, e a textura da Pokébola.

23. **Pokemon**
    Estrutura que define as propriedades de um Pokémon, incluindo nome, texturas (para as ações de ataque, defesa, etc.), e a cor associada.

24. **PokemonStats**
    Estrutura que guarda as estatísticas do Pokémon, como pontos de vida (HP), ataque e defesa.

25. **PlayerState**
    Estrutura que representa o estado atual de um jogador, como o Pokémon selecionado, a ação que ele está realizando, e um contador de frames para sincronizar animações.

26. **GameScreen**
    Enumeração que define os diferentes estados ou telas do jogo (menu principal, seleção de Pokémon, batalha, etc.).

27. **PokemonAction**
    Enumeração que define as possíveis ações de um Pokémon (parado, atacando, recebendo ataque).

28. **Lógica do jogo e efeitos visuais: WindowShouldClose**
    Verifica se a janela do jogo deve ser fechada, encerrando o loop principal do programa.

29. **UpdateMusicStream**
    Atualiza o estado da música de streaming, garantindo sua reprodução contínua.

30. **DrawTextureEx**
    Desenha uma textura em uma posição especificada com opções como escala e rotação.

31. **MeasureText**
    Calcula a largura de um texto em pixels para centralização ou dimensionamento.

32. **GetRandomValue**
    Gera um número aleatório entre dois valores inteiros especificados.

33. **DrawCircleV**
    Desenha um círculo em uma posição especificada, usando um vetor para coordenadas.

34. **IsKeyPressed**
    Detecta se uma tecla específica foi pressionada.

35. **PlaySound**
    Reproduz um som previamente carregado.

36. **SetTargetFPS**
    Define o número máximo de quadros por segundo que o jogo tentará renderizar.

37. **Controle de Piscar isBlinking = !isBlinking;**

    Alterna o estado da variável isBlinking, que controla o efeito de destaque visual piscante em elementos selecionados.

2. Controle de Telas (Blocos switch)

Tela do Menu Principal

38. **Bloco SCREEN_MAIN_MENU**

    Incrementa o contador de quadros (frameCounter) para animar o GIF de fundo.

    Usa UpdateTexture para atualizar o quadro do GIF com base no contador.

    Ao pressionar KEY_ENTER, muda para a tela de seleção do Pokémon do jogador 1 (SCREEN_POKEMON_SELECTION_PLAYER1).

Seleção de Pokémon

39. **Blocos SCREEN_POKEMON_SELECTION_PLAYER1 e SCREEN_POKEMON_SELECTION_PLAYER2**

    Permitem navegar entre os Pokémon disponíveis usando as teclas KEY_RIGHT e KEY_LEFT.

    Ao pressionar KEY_ENTER:

    Para o jogador 1, salva o Pokémon selecionado em player1State.selectedPokemon e muda para SCREEN_POKEMON_SELECTION_PLAYER2.

    Para o jogador 2, salva o Pokémon selecionado em player2State.selectedPokemon e muda para SCREEN_SCENARIO_SELECTION.

Seleção de Cenário

40. **Bloco SCREEN_SCENARIO_SELECTION**

    Permite navegar entre cenários com KEY_RIGHT e KEY_LEFT.

    Ao pressionar KEY_ENTER, muda para a tela de batalha (SCREEN_BATTLE).

41. **Tela de Batalha, Bloco SCREEN_BATTLE**

Exibe o cenário e os Pokémon selecionados pelos jogadores.

Controla as ações dos jogadores:

Ataques: KEY_A (Jogador 1) e KEY_J (Jogador 2).

Defesa: KEY_D (Jogador 1) e KEY_L (Jogador 2).

Aplica lógica de combate:

Reduz a vida (hp) do adversário ou aumenta a defesa do jogador atual.

Verifica o fim da batalha com base nos pontos de vida (hp) dos jogadores:

Exibe a mensagem de vitória e toca a música de vitória.

Permite transição para a tela de créditos ao pressionar Enter.

Tela de Créditos, Bloco SCREEN_CREDITS

Exibe uma animação de créditos.

Atualiza os quadros do GIF com UpdateTexture.

3. Funções Auxiliares

Funções Gráficas:

42. **DrawScaledTexture**

Desenha texturas dimensionadas para se ajustar à resolução da tela.

43. **DrawCenteredText**

Centraliza e exibe textos na tela com diferentes tamanhos e cores.

44. **DrawRectangle**

Desenha barras de vida na tela para representar o estado dos jogadores.

45. **DrawAttackEffect**

Cria um efeito visual para representar os ataques de um jogador.

Efeitos Visuais:

46. **FlashPokemonPlayer1 e FlashPokemonPlayer2**

Criam um efeito de piscada nos Pokémon dos jogadores durante a batalha, alternando entre visível e invisível.

Ações dos Jogadores:

47. **HandlePlayerActions**

    Gerencia as entradas do teclado para definir as ações dos jogadores, como ataque e defesa.

    Lógica de Combate

48. **AtaqueJogador1 e AtaqueJogador2**

    Calculam o dano causado ao adversário e aplicam à vida (hp) com base nos atributos de ataque e defesa.

Cálculo de Dano no Bloco da Batalha:

49. **int damage = playerXStats.attack - playerYStats.defense / 2;**

Fórmula usada para calcular o dano causado ao oponente, considerando ataque e defesa.

50. **Descarregamento de Recursos**

51. **Funções de Descarregamento**

52. **UnloadMusicStream, UnloadTexture, UnloadSound**: Liberam os recursos de mídia (músicas, texturas, sons) carregados durante o jogo.

53. Iterações (for) descarregam texturas relacionadas a Pokémon e cenários.

Outras Funções:

54. **StopMusicStream**

55. Para a reprodução da música de fundo atual.

56. **PlaySound**

57. Reproduz efeitos sonoros, como ataque ou vitória.

58. **UpdateTexture**

59. Atualiza texturas animadas (GIFs) com base no contador de quadros.

Carregamento de recursos:

60. **Cenários e Pokébolas**
    São definidos arrays contendo múltiplos cenários e Pokébolas, cada um com suas respectivas propriedades (nomes, texturas e cores).

61. **Pokémon**
    É definido um array contendo diferentes Pokémon, cada um com suas texturas para diferentes estados (parado, atacando e recebendo ataque) e uma cor associada.

Controle do jogo:

62. **player1State e player2State**
    Inicializam o estado de ambos os jogadores com o Pokémon selecionado e uma ação padrão de "parado".
63. **Transições de telas**
    São controladas por uma variável currentScreen, que muda de acordo com as interações do jogador.
64. **srand(time(NULL))**
    Inicializa o gerador de números aleatórios com base no tempo atual, garantindo resultados diferentes a cada execução.
65. **blinkCounter**
    Variável usada para criar efeitos visuais, como piscadas nos elementos visuais do jogo.

Efeitos finais:

66. **flashCounterPlayer1 e flashCounterPlayer2**
    Contadores usados para sincronizar o efeito de piscar dos Pokémon durante a batalha.
67. **Mensagens de batalha**
    Variáveis como battleMessage, battleMessage1 e battleMessage2 armazenam mensagens que serão exibidas na tela para orientar os jogadores.
68. **Controle de quadros (FPS)**
    O jogo é configurado para rodar a 60 FPS com SetTargetFPS, que é um valor ideal para jogos.
69. **Finalização de ações**

Ao final do contador de frames, as ações dos jogadores são revertidas para o estado padrão ("parado").

70. **Lógica da vitória**

Embora não explicitado diretamente no trecho, o jogo finaliza a batalha ao detectar que os pontos de vida (HP) de um jogador chegam a zero.

71. **Conformidade com a Raylib**:

Uso consistente de métodos padrão, como BeginDrawing() e EndDrawing() para manipulação de gráficos.

Descrição Geral do Código:

A lógica principal foi dividida em estados representados por telas (menus, seleção de personagens, seleção de cenários, batalhas em campo no jogo), utilizando estruturas de controle como switch para transitar entre elas.

Funções específicas tratam eventos de interação (ex.: teclas pressionadas), lógica de combate e renderização gráfica.

Esboço de Tópicos da Documentação

1. **Introdução**
   - Objetivos do Projeto.
   - Escolha do Tema e Justificativa.
2. **Planejamento e Equipe**
   - Distribuição de Tarefas entre os Integrantes.
   - Cronograma de Desenvolvimento.
3. **Ferramentas Utilizadas**
   - Recursos Externos.
   - Biblioteca Raylib: Principais Componentes e Vantagens.
4. **Lógica de Implementação**
   - Estrutura do Código.
   - Funções e Responsabilidades.
   - Fluxo de Execução do Jogo.
5. **Desafios e Soluções**
   - Problemas Enfrentados.
   - Escolhas Técnicas e Justificativas.
6. **Resultados Finais**
   - Avaliação do Projeto.
   - Pontos Positivos e Oportunidades de Melhoria.

Ferramentas a Serem Utilizadas

1. **Raylib:**
    - Renderização gráfica (2D e 3D).
    - Controle de áudio.
    - Entrada de usuário (teclado e mouse).
2. **Ferramentas de Suporte:**
    - **CapCut**: Edição de GIFs animados para o menu e créditos.
    - **Photoshop Pro**: Design de texturas e elementos visuais.
    - **VSCode**: IDE principal para desenvolvimento com extensões especializadas.
3. **Extensões de VSCode:**
    - Melhoram o suporte à escrita e compilação em C/C++.
    - Automatizam processos de build e debug.

Gerenciamento

A equipe foi composta por cinco integrantes, com papéis definidos:

- **Matheus** e **Lucas**: Responsáveis pela lógica de programação e desenvolvimento completo (front e back-end).
- **Kayron**: Principal responsável pelo design, pesquisa, documentação e implementação de soluções visuais.
- **Erick** e **Vinicius**: Focados em pesquisa e documentação do projeto.

O gerenciamento do tempo foi um desafio, com um prazo curto de uma semana. Apesar disso, foi possível concluir um protótipo funcional e agradável. A falta de comprometimento de dois integrantes dificultou o alcance de algumas metas mais ambiciosas.

Prazo e Resultados Finais

O projeto foi concluído em sete dias, atendendo ao prazo estipulado. O resultado final foi um jogo funcional que cumpre os objetivos iniciais, mas que deixa margem para melhorias. Ideias como animações mais dinâmicas e mecanismos de batalha para o oponente não foram implementados devido ao tempo limitado e à falta de colaboração de certos membros do grupo. O jogo alcançou os objetivos propostos, com uma mecânica simples e acessível, focada na batalha Pokémon, utilizando a biblioteca Raylib para a parte gráfica e interativa.

Entretanto, o jogo conseguiu demonstrar os conceitos de programação e design visual aprendidos, inspirando outras turmas a explorar além do material básico, incentivando a criatividade e o aprendizado contínuo.

A escolha das ferramentas, como o Visual Studio Code e a Raylib, foi eficaz para o cumprimento das tarefas de implementação. Embora o resultado final tenha sido satisfatório, há oportunidades de melhoria, principalmente na parte de dinamismo e interação do jogo. O projeto mostrou a importância de uma boa organização e da colaboração dentro de uma equipe, além de destacar a necessidade de um comprometimento contínuo para alcançar os melhores resultados.