
Relatório Técnico: Jogo 2D "Cavaleiro vs Slime"

Projeto: Desenvolvimento de Jogo 2D em HTML5 Canvas e JavaScript
Tema: Cavaleiro vs Slime
Desenvolvedor(es): Marco Antônio Clemente Ribeiro Guedes/ Miguel Oscar Silva Rodrigues De Souza
Período de Desenvolvimento: Outubro de 2025
Disciplina: Desenvolvimento de Jogos Digitais
Professor: Christien Lana Rachid

---

1. Introdução

Este relatório detalha o processo de desenvolvimento do jogo 2D de plataforma "Cavaleiro vs Slime", criado utilizando HTML5 Canvas e JavaScript puro. O projeto teve como objetivo explorar mecânicas clássicas do género, como movimento de personagem, colisões, animações baseadas em spritesheets, inimigos simples, itens colecionáveis e múltiplos níveis. Uma característica central deste desenvolvimento foi a **colaboração intensiva com Inteligências Artificiais (LLMs)**, especificamente o Gemini Pro, que auxiliou em diversas etapas, desde a ideação até a depuração final.

---

2. Processo de Desenvolvimento e Ferramentas

O desenvolvimento seguiu uma abordagem iterativa, começando com um esqueleto básico e adicionando funcionalidades progressivamente. As principais ferramentas foram:

- **Linguagens:** HTML5, CSS3, JavaScript (ES6+).
- **Ambiente:** Navegador Web (Opera GX), Editor de Código (VS Code).
- **Assets:** Spritesheets para o cavaleiro, slime, itens, tilesets de cenário e efeitos sonoros foram obtidos de fontes como itch.io e OpenGameArt.
- **Inteligências Artificiais:**
  - Gemini Pro(utilizada para a maior parte do desenvolvimento e depuração complexa)
  - Claude

---

3. Integração com IA (LLMs)

As LLMs foram parceiras essenciais durante todo o projeto. A interação evoluiu ao longo do tempo:

Fase Inicial (Prompts Estruturados):

No início, utilizei prompts mais detalhados e estruturados para gerar a base do código e as

mecânicas iniciais. Exemplos de prompts incluem:

- **Ideação:** "Atue como desenvolvedor de jogos 2D em HTML5 Canvas. Sugira 3 ideias de jogo no tema 'Cavaleiros com Slime', cada uma com: mecânicas centrais, entidades, estados do jogo, eventos de teclado, uso de paralaxe, colisão, spritesheet/clipping e disparo. Inclua uma lista de tarefas priorizadas (MVP → Polimento)."
- **Esqueleto do Código:** "Crie um projeto base de jogo 2D em Canvas com: index.html simples; style.css básico; main.js contendo: loop de animação com requestAnimationFrame, funções update() e draw(), controle de teclado, cenário com paralaxe, estrutura de entidades, sistema de colisão AABB. Inclua comentários."
- **Implementação de Spritesheet:** "Dado um spritesheet com {larguraFrame}x{alturaFrame} e {cols} colunas × {rows} linhas, implemente a animação do player (idle, run, shoot) no Canvas. Utilize drawImage(...) para clipping. Adicione controle de taxa de quadros."
- **Sistema de Disparo:** "Implemente um sistema de disparo: array de balas, criação ao pressionar tecla, velocidade, remoção. Adicione colisão bala vs inimigo (AABB) e pontuação no HUD."

Fase de Desenvolvimento e Depuração (Prompts Iterativos):

À medida que o jogo se tornava mais complexo, os prompts passaram a ser mais focados em adicionar funcionalidades específicas ou corrigir bugs:

- **Adição de Funcionalidades:** "Modifique a classe Player para ter um estado 'RUNNING' após segurar a tecla de movimento por X segundos.", "Implemente um sistema de múltiplos níveis usando diferentes tilesets.", "Adicione efeitos sonoros para pulo, ataque e música de fundo.", "Crie um menu inicial e uma opção para reiniciar o jogo."
- **Correção de Bugs (Exemplos das nossas conversas):** "Ele está preso na plataforma agora", "A animação do guerreiro continua bugada [...] ele está colocando no walk uma imagem que não deveria estar lá", "O herói só está dando um ataque e parando [...] e está travando na animação de dano", "O fundo não está se repetindo corretamente."

Impressões sobre a Colaboração com IA:

Achei a colaboração com a IA **extremamente útil**, especialmente para gerar rapidamente estruturas de código e sugerir abordagens para novas mecânicas. No entanto, o processo **deu bastante trabalho**. Depurar o código gerado pela IA, especialmente para problemas complexos como física de colisão e gestão de estado de animação, exigiu múltiplas iterações, explicações detalhadas dos bugs (incluindo o envio de imagens) e ajustes manuais. A IA não acertava sempre à primeira, e era necessário guiar o processo de correção passo a passo.

---

4. Vantagens da IA Paga (Sem Limite) vs. Gratuita (Com Limite)

Durante o desenvolvimento, utilizei tanto uma versão gratuita quanto uma versão paga

(Gemini Pro) de uma LLM. A diferença foi notável e impactou significativamente a produtividade:

- **Versão Gratuita:** Apresentava limites de uso (número de mensagens por hora/dia). Em momentos de depuração intensa, onde são necessárias muitas interações rápidas para testar soluções, esses limites eram **frustrantes** e **interrompiam o fluxo de trabalho**. Tinha que esperar ou alternar entre IAs, o que atrasava o progresso. Além disso, por vezes sentia que as respostas podiam ser mais lentas ou o modelo menos capaz de lidar com a complexidade crescente do código.
- **Versão Paga:** A **ausência de limites de uso** foi a maior vantagem. Pude iterar rapidamente na depuração dos bugs mais difíceis (colisão, animação) sem interrupções. Enviar o código completo, descrever o bug, receber uma sugestão, testar, reportar o novo problema, tudo de forma contínua. Isso foi **crucial** para resolver os problemas mais teimosos. Senti também que as respostas eram geralmente mais rápidas e, possivelmente, o modelo da versão paga era mais apto a compreender o contexto complexo de um ficheiro de código extenso e a gerar soluções mais precisas. A capacidade de enviar ficheiros e analisar imagens diretamente na conversa também foi um diferencial importante para diagnosticar bugs visuais.

Em resumo, a versão paga permitiu um ciclo de desenvolvimento e depuração muito mais ágil e eficiente, especialmente crucial para um projeto com prazo definido e com desafios técnicos inesperados.

---

5. Desafios Técnicos Encontrados

- **Física de Colisão:** Este foi, sem dúvida, o maior desafio. Implementar um sistema de colisão robusto que impedisse o jogador e os inimigos de atravessarem plataformas ou ficarem presos nos cantos exigiu várias refatorações. A solução final envolveu a implementação de uma verificação de colisão por separação de eixos (primeiro horizontal, depois vertical) gerida centralmente na classe Game.
- **Gestão de Estado de Animação:** Corrigir bugs visuais como o "efeito fantasma" (mistura de frames de animações diferentes) e garantir que as animações de ataque e dano não travassem o personagem exigiu um ajuste fino na classe AnimationController e na lógica de transição de estados na classe Player.
- **Renderização de Sprites:** Problemas iniciais com sprites desfocados ou mal recortados foram resolvidos garantindo o uso de coordenadas inteiras no desenho (Math.floor) e ajustando corretamente o frameWidth/frameHeight no AnimationController para corresponder aos assets originais.
- **Integração de Áudio:** Implementar um sistema simples para tocar sons em eventos específicos (pulo, ataque) e gerir a música de fundo em loop.

---

6. Resultados e Estado Atual do Jogo

O jogo "Cavaleiro vs Slime" encontra-se num estado funcional, implementando as seguintes características:

- Menu inicial e ecrãs de Game Over/Vitória com opção de reiniciar.
- Múltiplos níveis com temas visuais distintos (floresta, deserto, noite) usando tilesets.
- Movimento completo do jogador (andar, correr, pular) com animações correspondentes e sprites que viram de acordo com a direção.
- Sistema de combate corpo a corpo com múltiplos sprites de ataque aleatórios e ataque em corrida.
- Ataque especial (bola de fogo) acionado após um número de inimigos derrotados.
- Inimigo Slime com IA simples (persegue o jogador pulando) e animação.
- Sistema de colisão funcional com plataformas.
- Itens colecionáveis (maçã para cura, estrela para boost) e obstáculos (espinhos).
- HUD exibindo vida, contagem de kills e carga do especial.
- Efeitos sonoros básicos para ataque, pulo do slime e ambiente.
- Efeitos visuais simples (partículas, screen shake, trail do boost).
- Morte por queda implementada.

---

7. Conclusão

O desenvolvimento do "Cavaleiro vs Slime" foi uma experiência de aprendizado valiosa, demonstrando a viabilidade e os desafios de construir um jogo 2D com JavaScript e Canvas. A colaboração com LLMs, especialmente a versão paga e sem limites, foi fundamental para acelerar o processo e superar obstáculos técnicos complexos, embora tenha exigido um esforço considerável de depuração e orientação por parte do desenvolvedor. O jogo atingiu os requisitos básicos propostos, estabelecendo uma base sólida para futuras expansões, como a adição de mais níveis, inimigos com IA mais complexa, bosses e polimento audiovisual.

---

8. Referências (LLMs Utilizadas)

- [Claude]
- [Gemini Pro]
