Estudo de técnicas para o desenvolvimento de jogos RPG de ação 2D com a aplicação de um estudo de caso: um jogo em C++ com a biblioteca SDL

Thalles S. M. Barros[1], **Eduardo M. Cruz**[1]

[1]Engenharia de Software – Instituto Federal do Paraná (IFPR) – Campus Paranavaí
Telefone: (44)3482-0113 – CEP: 87703-536 – Paranavaí – PR – Brazil

thallessorrilha@gmail.com, eduardo.cruz@ifpr.edu.br

Abstract. *Game development combines Software Engineering challenges with domain-specific obstacles like physics and memory management. Specifically, 2D top-down action RPGs require a solid theoretical and technical foundation. This work investigates development techniques for this genre, focusing on Object-Oriented Programming (OOP) and the Game Loop architecture. To validate these concepts, a case study was conducted: developing the game "Insolitus" using C++ and the SDL library. Finally, the game's quality is assessed through a usability evaluation based on Nielsen's Heuristics adapted for electronic games.*

Resumo. *O desenvolvimento de jogos une desafios da Engenharia de Software a obstáculos específicos do domínio, como física e gerenciamento de memória. Neste contexto, RPGs de ação 2D top-down exigem um sólido embasamento teórico e técnico. Este trabalho pesquisa as técnicas de desenvolvimento aplicadas a esse gênero, focando na Programação Orientada a Objetos (POO) e na arquitetura Game Loop. Para aplicar os conceitos, conduziu-se um estudo de caso: o desenvolvimento do jogo "Insolitus" utilizando C++ e a biblioteca SDL. Para a avaliação da qualidade do jogo, foi aplicada uma avaliação heurística de usabilidade baseada nas Heurísticas de Nielsen adaptadas para jogos eletrônicos.*

1. Introdução

Apesar de aparentarem ser mero entretenimento, os jogos constituem projetos complexos na Engenharia de Software por demandarem soluções específicas, como motores de jogo, gestão de memória, físicas e colisões. Para esse fim, a linguagem C++ sobressai-se pela alta performance e manipulação de baixo nível [Nystrom 2014], frequentemente aliada à biblioteca SDL (*Simple DirectMedia Layer*) para acesso direto a gráficos, som e entradas [Simple DirectMedia Layer 2026].

Dentre os gêneros existentes, o RPG (*Role Playing Game*) de ação, exemplificado pela franquia *The Legend of Zelda* [Nintendo 1986], combina combate, progressão de personagens, narrativa e exploração. Contudo, impõe desafios técnicos particulares, a exemplo da ordenação gráfica via algoritmo *y sort* [Eriksson 2011]. Nesse cenário, investigar técnicas de desenvolvimento adequadas é fundamental para nortear os programadores e garantir a qualidade do software.

Embora existam trabalhos sobre padrões de projeto, RPGs e C++ com SDL de forma ampla, observa-se uma lacuna científica quanto ao desenvolvimento prático de um RPG de ação 2D *top-down* unindo C++ e SDL, com materiais frequentemente pagos ou pouco aprofundados. Além disso, a expansão do mercado nacional, que registrou um salto de 200 empresas em 2014 para 1.042 em 2023, atingindo o 10º lugar no ranking mundial [ABRAGAMES 2024], evidencia a relevância deste trabalho para a capacitação profissional.

Diante disso, o objetivo geral deste trabalho é estudar técnicas de desenvolvimento para jogos RPG de ação 2D por meio de um estudo de caso prático em C++ e SDL, validando-o via avaliação de usabilidade baseada nas Heurísticas de Nielsen adaptadas a jogos.

Os objetivos específicos compreendem:

1. Proporcionar um maior entendimento das técnicas empregadas no desenvolvimento de jogos.
2. Desenvolver o jogo aplicando as técnicas investigadas durante a pesquisa.
3. Mensurar quantitativamente e qualitativamente os aspectos do estudo de caso desenvolvido.

2. Trabalhos Relacionados

Para fundamentar esta pesquisa, realizou-se uma busca no Google Acadêmico por trabalhos correlatos utilizando as palavras-chave “Jogos”, “C++”, “SDL”, “RPG de ação” e “2D top-down”, identificando-se artigos focados em implementações práticas e estudos de caso aplicados.

2.1. Explorando C++ e SDL na Criação de um Jogo 2D do Gênero Metroidvania [Guimarães Neto 2025]

O trabalho investigou as vantagens e complexidades de se construir um motor próprio frente ao uso de soluções de terceiros por meio do desenvolvimento de um jogo Metroidvania em C++ com SDL. A avaliação com usuários validou a liberdade de customização proporcionada por engines próprias, apesar dos desafios técnicos como simulação de física. Sua relevância reside no relato dessa experiência de desenvolvimento e na disponibilização pública de seu repositório no GitHub para embasamento de soluções.

2.2. Desenvolvimento de jogo digital de RPG [Castelo Branco 2022]

Os autores desenvolveram um protótipo de RPG 2D na engine Construct guiado pelo framework Scrum, requisitos e diagramas UML. O sistema foi documentado pela metodologia MDA (Mecânicas, Dinâmica e Estética) para classificar os componentes do jogo. O estudo contribui para o presente trabalho ao explicitar componentes e dinâmicas fundamentais do gênero RPG.

2.3. Estudo de Técnicas de Implementação de Jogos 2D [Oliveira 2024]

Trata-se de um levantamento bibliográfico das principais técnicas de desenvolvimento de jogos, abordando escolha de motores, POO, arquitetura de software, física e arte, aplicado empiricamente a um jogo de cartas 2D em C++ e SDL. Contribui ao mapear teoricamente e aplicar na prática as técnicas de programação na mesma pilha tecnológica adotada nesta pesquisa.

2.4. Desenvolvimento de Archero Clone: Um Jogo de Ação 2D em C++ utilizando SDL2 [Santos 2024]

O artigo relata a engenharia reversa de um jogo de ação 2D em C++ e SDL2. Os autores detalham problemas de carregamento redundante e vazamento de memória, solucionados pelo uso de ponteiros inteligentes (`unique_ptr`). A pesquisa relaciona-se ao presente trabalho por apresentar soluções práticas para desafios comuns de gerenciamento de memória em C++/SDL.

2.5. Tower Building: Projeto e Desenvolvimento de um Jogo em C++ [Gauze 2024]

A pesquisa analisa o desenvolvimento de um jogo em C++ com SDL empregando Arquitetura Hexagonal e princípios SOLID para mitigar acoplamento e organizar o paradigma orientado a objetos. Sua relevância consiste na aplicação direta de padrões e arquiteturas formais de Engenharia de Software ao desenvolvimento com C++ e SDL.

2.6. Síntese dos Trabalhos Relacionados

A análise da literatura evidenciou as tecnologias e metodologias empregadas na área, com repositórios e relatos de problemas que auxiliam na implementação prática. Assim como Guimarães Neto, Oliveira, Santos e Gauze, este trabalho adota C++, SDL e engine própria. O gênero RPG alinha-se ao estudo de Castelo Branco, contudo, no âmbito de arquitetura de software, o presente projeto limita-se ao uso de POO e *Game Loop*, diferindo das abordagens mais complexas como a Arquitetura Hexagonal adotada por Gauze.

3. Estudo de Caso: Desenvolvimento do jogo “Insolitus”

As técnicas aprendidas ao longo dessa pesquisa são fundamentais para o desenvolvimento de jogos RPG de ação 2D *top-down*. E como forma de tornar esses conceitos mais tangíveis, houve a aplicação de um estudo de caso, que consiste no desenvolvimento de um jogo eletrônico do gênero RPG de ação 2D, chamado “Insolitus”.

Ao longo dessa seção será apresentado esse estudo de caso, discorrendo sobre suas características gerais, a metodologia utilizada para o desenvolvimento, e detalhes da implementação.

3.1. Descrição

“Insolitus” significa “estranho/incomum” em latim, e se refere ao tema do jogo, que passa uma sensação de mistério e aborda o sobrenatural. A história do jogo, começa quando o protagonista (personagem jogável) vê a sua vila sofrendo de acontecimentos sobrenaturais, e decide sair de sua casa para investigar. Ao longo de sua jornada ele se depara com personagens e criaturas que estão presentes no imaginário brasileiro, seja no folclore ou em lendas urbanas.

O jogo em questão se caracteriza pela progressão de *status* do personagem, coletamento de itens e desenvolvimento de uma histórica envolvente, os chamados jogos RPG. A constituição do mundo e cenários onde o jogador interage é caracterizada por 2 dimensões (norte/sul e oeste/leste), os chamados jogos 2D. A visualização desse mundo pelo jogador é tida na parte superior do mundo, formando como que um mapa visto de cima, permitindo o jogador andar nas direções norte/sul e leste/oeste, os chamados jogos

topdown. O jogo possui uma dinâmica de ação, baseada em combate em tempo real do jogador com os inimigos, por isso é considerado um jogo de ação.

Em geral, esse jogo se assemelha com os jogos da franquia *The Legend of Zelda* [Nintendo 1986], como o jogo de mesmo nome lançado para o console *Famicom* em 1986, como pode ser visto na Figura 1. Este, é um pioneiro em ação e aventura cuja dinâmica central se baseia na exploração livre e não-linear de um vasto mundo aberto. O jogo instiga a intuição do jogador ao esconder segredos em lugares ocultos, atrelando a progressão do jogador à descoberta e utilização de novos itens. A mecânica de *gameplay* mescla combate tático em quatro direções na perspectiva *top-down* com o gerenciamento de recursos. Jogos que reúnem essas características em geral, são chamados de jogos 'zelda-like' ("como *Zelda*", em tradução livre).

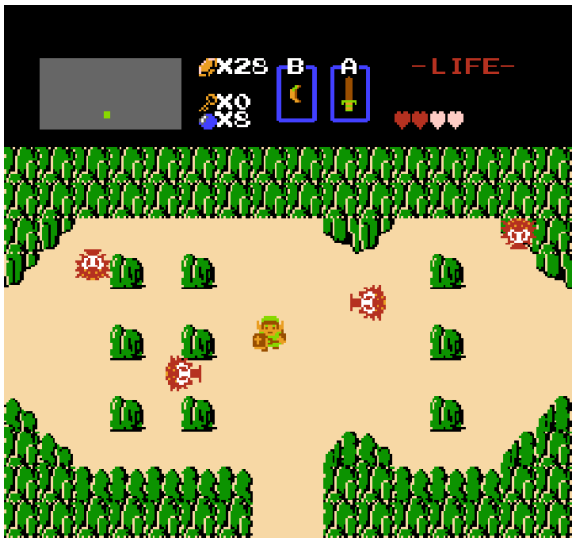


Figura 1. Cena do jogo *The Legend of Zelda* (1986).

Algumas das mecânicas presentes no jogo *Insolitus* são a física (força, atrito, aceleração, velocidade); as colisões (colisão de objetos com o mapa e objetos entre si); sistema de combate estilo RPG de ação, que permite com que o jogador controle seu personagem, ataque os inimigos e receba dano; e sistema de mapas, em que o *player* é capaz de percorrer uma cena e alterar entre cenas.

Uma importante mecânica presente em jogos RPG de ação é o sistema de combate. No jogo, o *player* é capaz de andar pelas 4 direções cardeais e desferir golpes, que causam dano nos inimigos. Por sua vez, os inimigos também são animados e causam dano em caso de toque com o *player*. Além disso, cada golpe desferido e cada dano recebido demoram um tempo para que possa ser ser executada uma outra ação novamente. Por fim, cada dano recebido causa uma força física contrária ao objeto que está desferindo o ataque, o chamado *knockback*.

3.2. Metodologia de Desenvolvimento

Em relação ao escopo, o jogo não será um produto comercial completo, mas um protótipo que busca incorporar as principais dinâmicas do gênero em um escopo reduzido, como a implementação de um único mundo ou fase. A implementação será conduzida com base no *framework* Scrum, por meio de ciclos iterativos e incrementais de desenvolvimento

[Schwaber and Sutherland 2014]. O GitHub foi usado com duas funções: Gerenciar o repositório do projeto e manter as histórias de usuário em um quadro Kanban. O repositório deste projeto pode ser visto no GitHub[1].

A linguagem C++ (versão standard 23) foi utilizada na programação do jogo, pois possibilita alto desempenho e eficiência de processamento. Junto disso, a biblioteca SDL, foi empregada para permitir de forma simplificada, acesso de baixo nível à recursos do sistema e hardware, como eventos de mouse e teclado, recursos de som e exibição gráfica. Outras bibliotecas que são utilizadas no jogo são a *my-lib* [Cruz 2026], para a manipulação de vetores matemáticos; e a *tinyxml* [Thomason 2026], que é utilizada para o carregamento de arquivos XML de mapas para o jogo.

Como paradigma de programação, a POO foi utilizada pois facilita a manipulação dos objetos do jogo, estabelecimento de uma hierarquia de classes, além de ser suportada pelo C++. Já em relação às arquiteturas que constituem o código, foi implementada apenas a de *Game Loop* em conjunto com o paradigma de POO, já que por se tratar do primeiro jogo desenvolvido pelo autor, estas técnicas proporcionam uma abordagem mais simples quando comparadas à implementação de arquiteturas de software mais elaboradas, como a Orientada à Eventos (EDA) e a Entidade-Componente-Sistema (ECS).

Para a programação foi utilizado o Visual Studio Code com o Copilot de apoio, de forma a auxiliar em questões pontuais, estudos iniciais ou tarefas repetitivas. A compilação será feita em Windows, por meio da plataforma MSYS2 UCRT64 que provê ferramentas e terminal para a compilação de códigos neste sistema operacional. O compilador utilizado foi o gcc (versão 16.1).

O processo de teste de software se baseou no uso do terminal do MSYS2 UCRT64, tanto para depurar a compilação do código quanto para exibir o valor de variáveis e ponteiros em tempo de execução. Dessa forma, foi possível descobrir erros e alertas na compilação, como variáveis não usadas, erros de semântica e falta de importações. Ademais, foi possível visualizar em tempo real o comportamento da lógica do jogo e da memória utilizada, permitindo aperfeiçoar o comportamento do jogo, descobrir vazamento de memória e visualizar endereço de ponteiros.

Os *assets* de jogos se referem à recursos do jogo, como *sprites* (desenhos gráficos), efeitos sonoros e música. No jogo em questão eles foram obtidos por meio da loja virtual de *assets*, *Itch.io* sendo o principal fornecedor, um conjunto chamado “Ninja Adventure”, que oferece uma variedade de recursos de forma gratuita (CC0 - domínio público) [PixelBoy and AAA 2021].

Já o mapa do jogo foi confeccionado por meio do software *Tiled*, que permite importar conjuntos de *tiles* (quadrados do jogo), os chamados *tilesets* e definir como será cada bloco das camadas do respectivo mapa. Um exemplo de uso dessa ferramenta pode ser visto na Figura 2. Juntamente com as camadas visuais, podem ser definidas também uma camada de colisão, que exibe quais blocos são colisíveis para os objetos; e a posição que esses objetos começam na renderização da cena.

[1] https://github.com/ThallesSorrilha/Cpp_Game

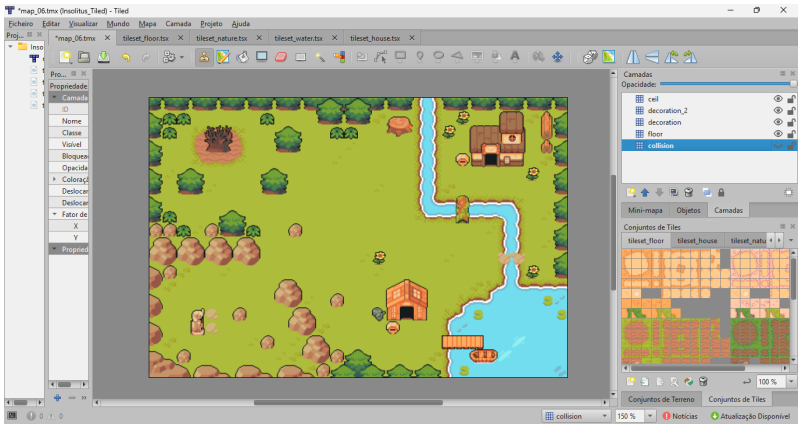


Figura 2. Exemplo de uso da ferramenta *Tiled*.

3.3. Implementação

Com relação ao planejamento do jogo, inicialmente foi esboçado um diagrama de classes para planejar a implementação das classes, como pode ser visto na Figura 3. Observando a estrutura de classes, pode notar o seguinte:

GameObject Representa os objetos do jogo, em geral, e suas classes derivadas.
Player Jogador principal.
Enemy Inimigo.
AttackObject Objeto de ataque.
TextureManager Classe utilitária que funciona como gerenciador de texturas.
TmxLoader Classe utilitária que atua como carregador de mapas em formato TMX.
GameManager Responsável pelo gerenciamento geral do jogo.
GameWorld Classe que representa um mundo no jogo.
TileMap Classe que representa o conjunto de blocos de um mapa.
Camera Controla a visualização da câmera sobre o *Player*.
ColliderBox Caixas de colisão presentes no jogo.
CollisionManager Administrador de colisão, responsável por verificar as *ColliderBox*.

4. Metodologia de Avaliação

Para a avaliação do estudo de caso, será feita uma avaliação heurística do jogo final. A avaliação se baseia nas 10 heurísticas de usabilidade de Nielsen [Nielsen 2020], porém, de forma adaptada ao contexto dos jogos, como pode ser visto no trabalho de [Desurvire and Wiberg 2009].

As heurísticas são diretrizes ou princípios que são considerados adequados para interfaces de usuário e experiência de usuário, considerando aspectos psicológicos [Nielsen 2020]. Nesse contexto, Nielsen definiu 10 princípios de usabilidade que se aplicam para softwares interativos, em geral. Já o trabalho de [Desurvire and Wiberg 2009], redefine essas heurísticas, considerando aspectos que são fundamentais para jogos eletrônicos, no geral.

Para a avaliação do jogo, foi feita ainda uma adaptação das heurísticas presentes no trabalho de [Desurvire and Wiberg 2009], levando em conta o escopo proposto para o jogo em questão. A avaliação elaborada consiste de um formulário, que busca analisar

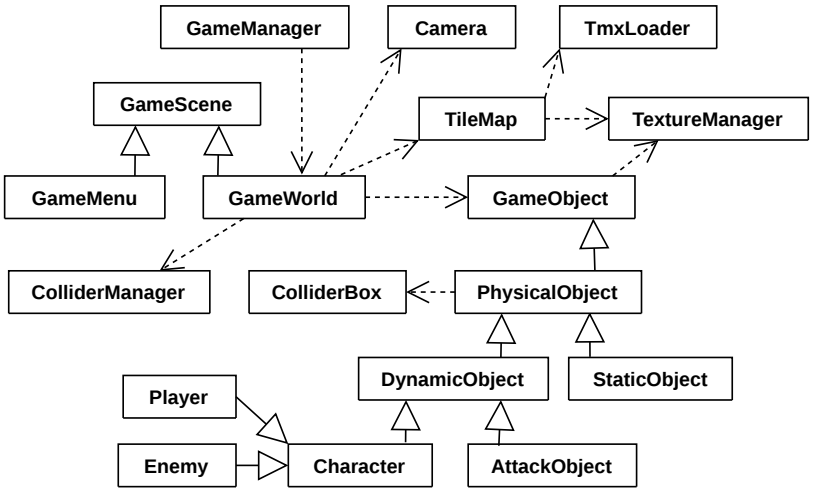


Figura 3. Diagrama de classes do jogo.

o jogo sob a ótica de cada uma das 16 heurísticas estabelecidas. Para cada uma, será atribuída uma nota de 0 a 10, junto de uma descrição qualitativa, explicitando o alinhamento da heurística com as características do jogo. As heurísticas que serão avaliadas podem ser vistas na Tabela 1.

5. Resultados Obtidos

5.1. O jogo

Em relação ao estudo de caso, foi desenvolvido o jogo *Insolitus*, que buscou capturar os principais conceitos de jogos RPG de ação 2D. Até o momento, o jogo possui implementação da estrutura de classes do jogo, arquitetura *Game Loop*, movimentação do personagem principal, simulação de física baseada em forças, aceleração e velocidade, aleatorização de movimento dos inimigos, sistema de combate, comando de ataque do *player*, recebimento de dano pelos personagens, exibição gráfica animada de personagens, exibição do mapa segundo camadas, além de *sprites* e músicas que se encontram prontos.

Esse jogo, tem como objetivo prático, eliminar todos os monstros do mapa e o chefão final. Para isso, o jogador deve explorar o mapa e atacar os inimigos, tomando cuidado para não sofrer dano e morrer. Ao derrotar 10 inimigos, será liberada uma passagem para enfrentar o chefão final.

Uma amostra do jogo pode ser vista no conjunto de cenas da figura 4a. A figura 4a mostra o início da jogatina, em que o jogador, um menino, surge em uma clareira na floresta. O único caminho livre é à frente, dessa forma, o jogador é instigado à testar o primeiro comando básico do jogo, o andar. A figura 4b, mostra o que acontece logo em seguida, em que o jogador se depara com uma área mais ampla, ao qual pode explorar e encontrar novos objetos como matinhos, ao qual pode cortar; e esqueletos, que são inimigos e causam dano ao jogador.

Além disso, também é possível infligir dano aos inimigos golpeando eles com um taco, por meio do comando de ataque, como pode ser visto na figura 4b. No jogo,

Tabela 1. Heurísticas de Avaliação do Jogo

Categoria	Heurística
Jogabilidade	Retenção do Jogador Desafio, Estratégia e Ritmo Consistência de Mundo do Jogo Objetivos do Jogo Variedade de Jogadores e Estilos de Jogo Percepção de Controle pelos Jogadores
Entretenimento e Imersão	Entretenimento Imersão
Usabilidade e Mecânicas do Jogo	Tutorial e Documentação Pontuação e status Jogo proporciona Feedback Terminologia Responsabilidade do Jogador Layout de Tela Navegação Prevenção de Erro

os esqueletos são inimigos menos inteligentes, que apenas andam aleatoriamente pelo mapa, enquanto os inimigos encapuzados de preto, são mais inteligentes, e conseguem perseguir o jogador, causando dano, e o levando à morte. Ao longo do jogatina, o jogador pode percorrer os cenários do jogo, como um campo (figura 4b), a vila (figura 4c), um cemitério (figura 4d) e um rochedo até chegar à área do chefão, onde haverá um embate final com o inimigo mais forte da história.

5.2. Avaliação Heurística

Para averiguar a eficácia das técnicas aplicadas sobre o jogo, foi conduzido uma avaliação heurística de usabilidade de Nielsen, considerando cada uma das heurísticas estabelecidas.

Retenção do Jogador — Nota: 4 / 10 O jogo apresenta dificuldades significante para prender a atenção do jogador no médio. As mecânicas básicas conseguem gerar um engajamento inicial momentâneo, mas a ausência de sistemas claros de progressão, recompensas contínuas ou fatores de rejogabilidade (como desbloqueáveis ou metas secundárias) faz com que a experiência se torne repetitiva rapidamente.

Desafio, Estratégia e Ritmo — Nota: 6 / 10 A curva de dificuldade é razoável, mas carece de refinamento. O jogo consegue alternar momentos de menor e maior tensão, porém o equilíbrio entre a habilidade exigida e as ferramentas oferecidas oscila. Em determinados momentos, o sucesso depende mais de repetição mecânica do que da formulação de estratégias elaboradas pelo jogador.

Consistência de Mundo do Jogo — Nota: 6 / 10 O universo do jogo apresenta uma coerência visual e temática razoável, com integração entre os elementos 2D e o cenário e permitindo que as regras do ambiente mantenham um padrão estável.

Apesar disso, um ponto negativo é a existência de uma única área, em que todos os objetos do jogo são renderizados. Isso pode fazer com que o jogador fique um pouco perdido por ser muito amplo.

Objetivos do Jogo — Nota: 4 / 10 Embora o objetivo principal do jogo ou da fase seja compreensível em linhas gerais, os objetivos secundários ou os passos intermediários necessários para alcançá-lo frequentemente carecem de clareza. O jogador por vezes se vê incerto sobre o que deve fazer a seguir ou qual é a utilidade direta de determinada ação no progresso do jogo.

Variedade de Jogadores e Estilos de Jogo — Nota: 3 / 10 O design do jogo é bastante rígido e linear, oferecendo pouca ou nenhuma margem para diferentes abordagens (ex.: furtiva vs. agressiva, exploração vs. velocidade). Não há opções significativas de customização, rotas alternativas ou flexibilidade de controles para comportar perfis variados de jogadores (casual, competitivo ou explorador).

Percepção de Controle pelos Jogadores — Nota: 7 / 10 A resposta aos comandos é satisfatória. O jogador sente que o personagem/avatar no espaço 2D responde de forma precisa e direta na maior parte do tempo, garantindo que acertos e erros sejam percebidos como consequência direta de suas próprias ações na interface.

Entretenimento — Nota: 3 / 10 O fator diversão está comprometido. A combinação de objetivos confusos, baixa retenção e repetição excessiva faz com que o *loop* principal de jogabilidade se torne monótono rapidamente, falhando em gerar a sensação de recompensa ou satisfação ao superar os obstáculos do jogo.

Imersão — Nota: 6 / 10 A capacidade de envolver o jogador é significante. Com uma harmonia entre os elementos visuais e o áudio de ambiente, o jogador consegue se concentrar no universo do jogo. A experiência atinge um nível satisfatório de absorção, embora detalhes pontuais de interface e ritmo ainda lembrem ocasionalmente o jogador da estrutura do sistema.

Tutorial e Documentação — Nota: 4 / 10 O aprendizado inicial ocorre de forma rasa. O jogo apresenta as mecânicas fundamentais por meio de textos e instruções pontuais pouco intuitivas no início, em vez de integrar o aprendizado organicamente ao *level design* (*learning by doing*). Faltam menus acessíveis de consulta de controles durante a partida.

Pontuação e Status — Nota: 5 / 10 A exibição das informações vitais no HUD alcançou um estado funcional. O jogador consegue consultar métricas como vida, pontuação e recursos de forma aceitável durante a partida, mas a hierarquia visual e o posicionamento das barras ainda carecem de refinamento para permitir leituras dinâmicas sem distrair da ação central em 2D.

Jogo Proporciona *Feedback* — Nota: 8 / 10 Ponto forte do jogo. O sistema responde muito bem às interações do usuário por meio de *feedbacks* visuais e/ou sonoros claros (ex.: animações de dano, efeitos de acerto, sons de coleta e transições de tela). O jogador raramente fica em dúvida se uma ação foi registrada pelo sistema.

Terminologia — Nota: 4 / 10 A linguagem nos menus e tutoriais é parcialmente uniformizada, pois apresenta inconsistências em nomenclaturas e iconografia entre telas diferentes, exigindo um pequeno esforço adicional do jogador para associar termos idênticos a funções ou recursos ao longo da navegação.

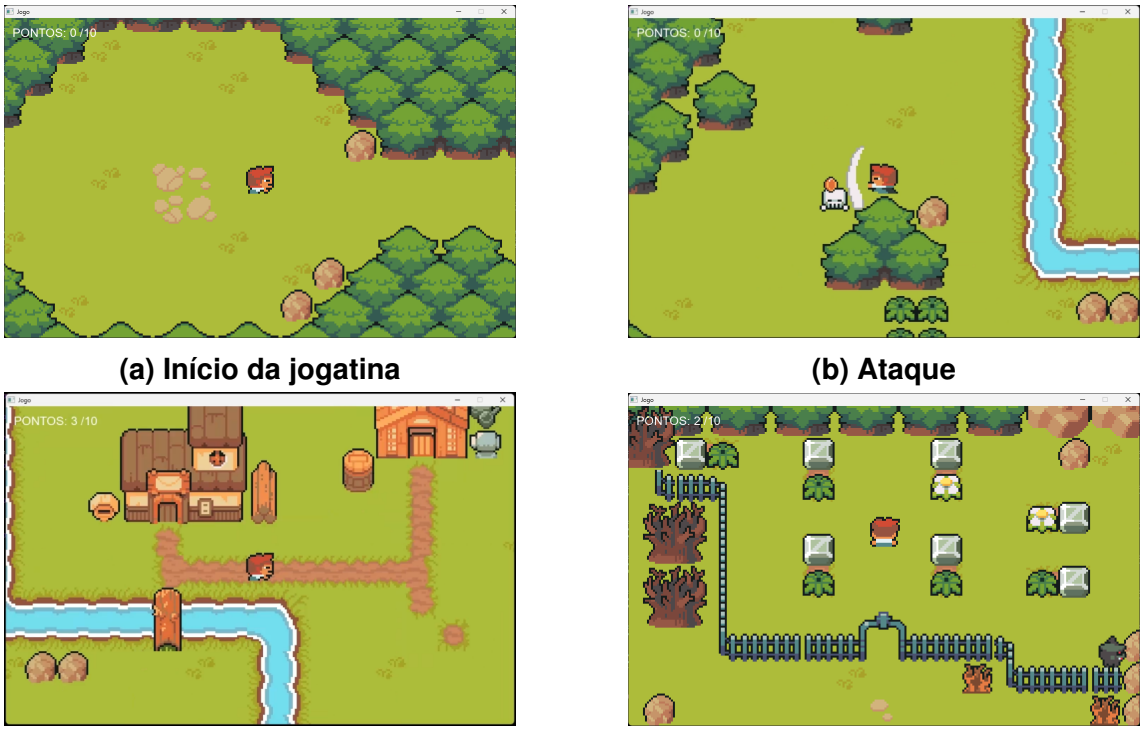
Responsabilidade do Jogador — Nota: 6 / 10 Quando o jogador erra (ex.: toma dano ou perde uma vida), ele consegue compreender razoavelmente que a falha foi sua e não um erro imprevisível do jogo. No entanto, em trechos com *hitbox* imprecisa

ou leitura de tela poluída, a sensação de injustiça pontual ainda pode ocorrer.

Layout de Tela — Nota: 4 / 10 A distribuição gráfica dos elementos de interface na tela carece de organização e hierarquia visual. O HUD e as janelas de menu poluem o campo de visão da ação 2D ou apresentam contraste inadequado, dificultando a leitura rápida das informações durante a jogabilidade.

Navegação — Nota: 4 / 10 A movimentação entre menus, telas de pause e transições de fase é pouco fluida e pouco intuitiva. A estrutura de menus exige cliques/botões excessivos para tarefas simples, e a ausência de atalhos claros dificulta a fluidez da experiência fora da ação principal.

Prevenção de Erro — Nota: 0 / 10 Ponto crítico e inexistente no jogo. O sistema não oferece nenhuma proteção contra ações acidentais e destrutivas do jogador (como sair do jogo sem salvar, reiniciar a fase por toque acidental, ou gastar recursos sem confirmação). A ausência total de diálogos de confirmação ou travas de segurança resulta em constante frustração.



(a) Início da jogatina **(b) Ataque**

(c) Cenário: Vila **(d) Cenário: Cemitério**

Figura 4. Jogo desenvolvido

6. Conclusão e Trabalhos Futuros

Este trabalho apresentou um estudo das principais técnicas empregadas no desenvolvimento de jogos RPG de ação 2D *top-down*, bem como da aplicação de um estudo de caso que consistiu no desenvolvimento de *Insolitus* - um jogo em C++ com a biblioteca SDL.

Os trabalhos correlatos estudados fornecem uma base para sintetizar o desenvolvimento de jogos, demonstrando metodologias que foram aplicadas para o estudo e aplicação de técnicas de desenvolvimento de jogos. Além disso, esses trabalhos relatam os

problemas que foram encontrados durante a implementação dos jogos e as soluções adotadas para resolvê-los. Dessa forma o presente trabalho conseguiu se apoiar na realização de estudos e práticas já validados.

A etapa de pesquisa trouxe as principais tecnologias e técnicas utilizadas no desenvolvimento de jogos 2D em geral, explorando as linguagens, bibliotecas, paradigmas e arquiteturas. Em breve ainda, serão adicionados mais tópicos sobre padrões de projeto para jogos e técnicas específicas de jogos RPG de ação 2D *top-down*.

O desenvolvimento do jogo empregou as metodologias definidas e as técnicas que foram estudadas, produzindo um protótipo jogável. A diagramação de classes permitiu planejar a estrutura geral do jogo antes da implementação em si. O seguimento da metodologia *Scrum*, junto do uso do quadro *Kanban* permitiu o monitoramento de histórias de usuário e entrega contínua de novas funcionalidades. A programação auxiliada pelo *Copilot*, de uma forma consciente se mostrou uma ferramenta importante para o desenvolvimento de um sistema complexo, como um jogo. O repositório no *GitHub* e uso de *Git* também possibilitaram o versionamento do código, facilitando a programação.

O resultado do jogo, até o momento, se demonstra como um protótipo jogável, com a implementação da lógica de batalha, movimentação, exibição de mundo e personagens do jogo. O código foi desenvolvido buscando evitar vazamento de memória e manter uma estrutura de classes condizente com a dinâmica do jogo.

Com tudo que foi desenvolvido até o momento, pode-se concluir que a aplicação das técnicas se demonstra satisfatória, trazendo benefícios práticos para o jogo final.

No atual momento, ainda estão pendentes alguns tópicos do trabalho, sendo eles: uma pesquisa sobre técnicas específicas aplicadas em jogos RPG de ação 2D *top-down*; implementação de algumas funcionalidades do jogo; e realização da avaliação heurística. Estes tópicos estão em desenvolvimento, e serão apresentados na próxima entrega.

Como trabalhos futuros, uma possível recomendação, seria a de aplicar uma arquitetura mais avançada, que permita uma maior componentização de classes e combinação entre elas de forma mais simples. Dada essa situação, a arquitetura ECS seria recomendada, possibilitando maior diversidade de tipos de objetos com maior flexibilidade e menos repetição de código.

Referências

ABRAGAMES (2024). *O jogo muito além dos games*. Associação Brasileira das Incubadoras de Jogos Digitais.

Castelo Branco, Y. d. S. (2022). Desenvolvimento de jogo digital de RPG. Technical report, Instituto Federal do Rio Grande do Norte (IFRN), Natal, RN.

Cruz, E. H. M. (2026). My-lib.

Desurvire, H. and Wiberg, C. (2009). Game usability heuristics (play) for evaluating and designing better games: The next iteration. In *International conference on online communities and social computing*, pages 557–566. Springer.

Eriksson, M. (2011). The history and future of computer RPG development. Trabalho de conclusão de curso (bacharelado em tecnologia da informação), Arcada University of Applied Sciences, Helsinki, FIN.

Gauze, Y. (2024). Tower building: Projeto e desenvolvimento de um jogo em C++. In *Anais da XI Semana de Tecnologia da Informação (SETIF)*, page 3, Paranavaí, PR. Instituto Federal do Paraná (IFPR).

Guimarães Neto, R. A. P. (2025). Desenvolvimento de um jogo metroidvania em C++ com SDL: Uma abordagem sem o uso de third-party engines. Technical report, Instituto Federal do Paraná (IFPR), Paranavaí, PR.

Nielsen, J. (2020). 10 usability heuristics for user interface design. Nielsen Norman Group. Atualizado em 15 de nov. de 2020.

Nintendo (1986). The legend of zelda. Jogo eletrônico. Plataforma: Nintendo Entertainment System (NES).

Nystrom, R. (2014). *Game Programming Patterns*. Genever Benning.

Oliveira, J. S. (2024). Estudo de técnicas de implementação de jogos 2D. Technical report, Instituto Federal do Paraná (IFPR), Paranavaí, PR.

PixelBoy and AAA (2021). Ninja pack. itch.io.

Santos, S. (2024). Desenvolvimento de Archero clone. In *Anais da XI Semana de Tecnologia da Informação (SETIF)*, page 3, Paranavaí, PR. Instituto Federal do Paraná (IFPR).

Schwaber, K. and Sutherland, J. (2014). *Um guia definitivo para o Scrum: as regras do jogo*. Scrum.Org / ScrumInc.

Simple DirectMedia Layer (2026). SDL version 2.0. Disponível em: https://www.libsdl.org/. Acessado em: 10 de maio de 2026.

Thomason, L. (2026). Tinyxml2.