Macri: Design e Desenvolvimento de um Jogo de Luta 2D

André Ricardo Zavan[1], **Iago Gonçalves de Meira**[1]

[1]Instituto Federal do Paraná (IFPR)
Paranavaí, PR – Brasil

andre.zavan@ifpr.edu.br, 20220009931@estudantes.ifpr.edu.br

Nos últimos anos, os jogos eletrônicos consolidaram-se como um dos principais mercados de produtos digitais, com forte impacto econômico e cultural em escala global. Em 2017, o Brasil ocupou a quarta posição mundial em número de vendas de jogos, segundo [Amélio 2018]. Embora essa pesquisa aponte uma discrepância entre volume de vendas e faturamento, além da percepção de desenvolvedores brasileiros de que políticas públicas protecionistas têm prejudicado o setor, o estudo reforça a relevância dos jogos eletrônicos tanto como mercado em expansão quanto como mídia cultural significativa.

Diante desse cenário promissor, é possível aplicar os conhecimentos adquiridos ao longo da graduação em Engenharia de Software para o desenvolvimento de um jogo de luta *2D*, integrando conceitos técnicos e acadêmicos em uma experiência prática. Os **jogos de luta *2D*** são característicos pela sua natureza competitiva e dinâmica desde os jogos de *arcade* [Ketonen 2016] até os dias atuais nos jogos de *consoles* e computadores pessoais (PC).

Experimentos semelhantes foram feitos anteriormente [Ferreira et al. 2011], buscando promover experiências didáticas através do desenvolvimento de jogos eletrônicos. Em concordância com esta proposta didática, um dos desafios deste projeto é criar uma arquitetura suficientemente robusta, extensível e de fácil manutenção, utilizando conhecimento de Interação Humano-Computador (IHC) e padrões arquiteturais comuns aplicados junto aos princípios de desenvolvimento SOLID (*Single Responsibility, Open/Closed, Liskov Substitution, Interface Segregation, e Dependency Inversion*) [Valente 2020].

A revisão bibliográfica abrangeu desde cálculos de colisões aplicados em formas geométricas [Mileff 2023] a princípios de animação em jogos de luta [Bahia et al. 2020] e estudos de IHC aplicados no desenvolvimento de jogos, que destacam a importância de equilibrar acessibilidade e desafio entre os jogadores [Barr et al. 2007]. O projeto tem como requisitos mínimos a presença de dois personagens jogáveis e um cenário de combate, acessíveis por menus de seleção e com suporte exclusivo para *multiplayer* local, devido à complexidade do desenvolvimento de inteligência artificial para combates. O jogo será destinado a dispositivos *desktop*, com suporte para teclado e *joystick*.

A linguagem C++ foi escolhida para o desenvolvimento, devido à sua ampla utilização em jogos e suporte a *engines* consagradas como *Unreal* e *Unity*. Foram adotadas as bibliotecas *SDL2* (*Single DirectMedia Layer* versão 2) e suas extensões *SDL2_image, SDL2_ttf e SDL2_mixer* para manipulação de áudio, entrada e gráficos [SDL Contributors 2025]. Além disso, integrou-se a biblioteca *JSON* (*Javascript Object Notation*)de *Niels Lohmann (Nlohmann JSON)*, facilitando o armazenamento e carregamento de dados estruturados.

O sistema conta com um gerente de eventos *Singleton* que se baseia no padrão de *software Observer* chamado *GameEventManager*, responsável pela tradução de eventos

SDL2 e pela comunicação entre diferentes classes que geralmente atuam como subsistemas capazes de gerenciar componentes. A Figura 1 mostra um exemplo desse comportamento através de um Diagrama de Sequências (DS-UML). A criação de um objeto *HealthComponent* chama o método *subscribe* do *GameEventManager* para observar *GameEvents* do tipo *DamageDealtEvent*, publicados pelo *CollisionSystem* com o método *publish* do *GameEventManager*. Realizada a publicação, o gerente chama uma função submetida pelo *HealthComponent* ao se inscrever, que no caso é a *onDamageDealtEvent*.

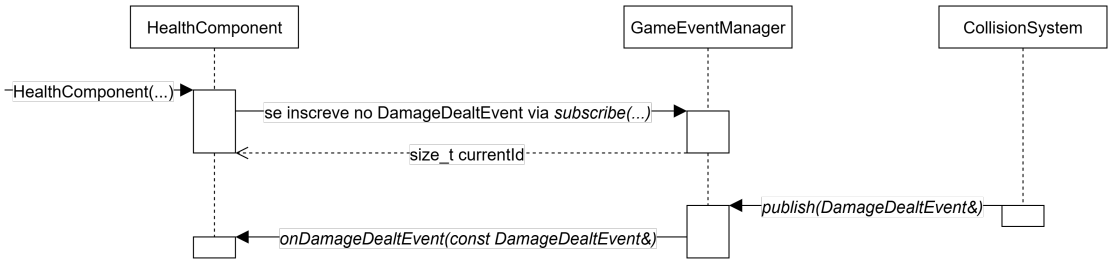

Figura 1. DS-UML - *HealthComponent* interagindo com *GameEventManager*

Os *subsistemas* do projeto têm ao menos uma classe *Singleton* chamada por um método estático *getInstance*. Como muitas dessas classes dependem de dependências externas, o Princípio de Inversão de Dependências (DIP) foi aplicado através da Injeção de Dependências em métodos *init* que recebem instâncias de classes concretas que utilizam os recursos *SDL2*. Muitos exemplos assim podem ser vistos na Figura 2. Esta técnica oferece os serviços das dependências *SDL2* de forma segura, desacoplada e com substituições de mínimo impacto.

```
void SDL2Application::init()
{
    std::cout << "Initializing SDL2Application..." << std::endl;
    try
    {
        Initializer::getInstance().init(new SDL2Initializer());
        GameEventTranslator::getInstance().init(new SDL2GameEventTranslator());
        Window::getInstance().init(new SDL2Window());
        Renderer::getInstance().init(new SDL2Renderer());
        ResourceManager::getInstance().init(new SDL2ResourceManager());
        InputSystem::getInstance().init(new SDL2InputSystem());

        /* Music and Sound Systems are still under development and not yet Singletons */
        this->musicSystem_ = std::make_unique<SDL2MusicSystem>();
        this->soundSystem_ = std::make_unique<SDL2SoundSystem>();

        Game::getInstance().init(new SDL2Game());
        std::cout << "SDL2Application initialized successfully." << std::endl;
    }
    catch (const std::exception& e)
    { std::cerr << "ERROR: SDL2Application initialization failed: " << e.what() << std::endl; }
}
```

Figura 2. SDL2Application::init() - Exemplos de Injeção de Dependência

Os próximos passos do desenvolvimento do projeto incluem finalizar o sistemas de animação e o carregamento de recursos, utilizando comandos de *input* enviados via

GameEventManager para coordenar os sistemas de colisão, física e animação. Com a finalização do motor básico do jogo, jogadores reais avaliarão o produto final em relação ao seu cumprimento com os requisitos de qualidade, desde a interação com a interface de seleção de personagens e fases até as lutas, o ponto mais importante do jogo no que diz respeito ao fator entretenimento.

Referências

Amélio, C. d. O. (2018). A indústria e o mercado de jogos digitais no brasil. *XVII SBGames, Foz do Iguaçu, Paraná, Brasil*, pages 1497–1506. Disponível em https://www.sbgames.org/sbgames2018/files/papers/IndustriaFull/188510.pdf. Acesso em 29 ago. 2025.

Bahia, S., Izolani, L. G., and Stein, M. (2020). 12 princípios de animação em jogos de luta. *Proceedings of SBGames 2020*. Disponível em https://www.sbgames.org/proceedings2020/ArtesDesignFull/209727.pdf. Acesso em 26 mar. 2025.

Barr, P., Noble, J., and Biddle, R. (2007). Video game values: Human–computer interaction and games. *Interacting with Computers*, 19(2):180–195. HCI Issues in Computer Games. Disponível em https://www.sciencedirect.com/science/article/pii/S0953543806001159. Acesso em 4 abr. 2025.

Ferreira, L. H., Pires, D. S., Dantas, M. S., and Lima, F. R. d. (2011). Produção de jogos digitais educativos por alunos do ensino superior. *Anais do Workshop sobre Educação em Computação (WEI)*. Disponível em https://sol.sbc.org.br/index.php/wei/article/download/3486/3445. Acesso em: 27 mar. 2025.

Ketonen, M. (2016). Designing a 2d fighting game. Thesis, Bachelor of Business Administration, Business Information Technology. Disponível em https://www.theseus.fi/bitstream/handle/10024/118514/Thesis_Miikka_Ketonen_KAT13PT.pdf?sequence=1. Acesso em 27 mar. 2025.

Mileff, P. (2023). Collision detection in 2d games. *Production Systems and Information Engineering*, 11(3):10. Disponível em https://ojs.uni-miskolc.hu/index.php/psaie/article/view/2167. Acesso em 24 mar. 2025.

SDL Contributors (2025). *Simple DirectMedia Layer Wiki*. SDL. Versão 2.30.4. Disponível em https://wiki.libsdl.org/. Acesso em 19 de set. 2025.

Valente, M. T. (2020). Engenharia de software moderna. *Princípios e práticas para desenvolvimento de software com produtividade*, 1(24). Disponível em https://engsoftmoderna.info/. Acesso em 25 mai. 2025.