UNIVERSIDADE FEDERAL DO CEARÁ

INSTITUTO UNIVERSIDADE VIRTUAL

CURSO DE GRADUAÇÃO EM SISTEMAS E MÍDIAS DIGITAIS

LUÍS EDUARDO REGIS DE OLIVEIRA

PROJETO E DESENVOLVIMENTO DO JOGO *HEXACHRONOS*: MESCLANDO *TURN-BASED STRATEGY* E *VISUAL NOVEL*

FORTALEZA

2022

LUÍS EDUARDO REGIS DE OLIVEIRA

PROJETO E DESENVOLVIMENTO DO JOGO *HEXACHRONOS*: MESCLANDO *TURN-BASED STRATEGY* E *VISUAL NOVEL*

Trabalho de Conclusão de Curso apresentado ao Curso de Graduação em Sistemas e Mídias Digitais do Instituto Universidade Virtual da Universidade Federal do Ceará, como requisito parcial à obtenção do grau de bacharel em Sistemas e Mídias Digitais.

Orientador: Prof. Dr. José Gilvan Rodrigues Maia

FORTALEZA

2022

Dados Internacionais de Catalogação na Publicação
Universidade Federal do Ceará
Sistema de Bibliotecas
Gerada automaticamente pelo módulo Catalog, mediante os dados fornecidos pelo(a) autor(a)

O48p Oliveira, Luís Eduardo Regis de.
Projeto e desenvolvimento do jogo hexachronos : mesclando turn-based strategy e visual novel / Luís Eduardo Regis de Oliveira. – 2022.
42 f. : il. color.

Trabalho de Conclusão de Curso (graduação) – Universidade Federal do Ceará, Instituto UFC Virtual, Curso de Sistemas e Mídias Digitais, Fortaleza, 2022.
Orientação: Prof. Dr. José Gilvan Rodrigues Maia.

1. Jogos eletrônicos. 2. Visual novel. 3. Turn-based Strategy. 4. Relato de Experiência. I. Título.
CDD 302.23

LUÍS EDUARDO REGIS DE OLIVEIRA

PROJETO E DESENVOLVIMENTO DO JOGO *HEXACHRONOS*: MESCLANDO *TURN-BASED STRATEGY* E *VISUAL NOVEL*

Trabalho de Conclusão de Curso apresentado ao Curso de Graduação em Sistemas e Mídias Digitais do Instituto Universidade Virtual da Universidade Federal do Ceará, como requisito parcial à obtenção do grau de bacharel em Sistemas e Mídias Digitais.

Aprovada em:

BANCA EXAMINADORA

Prof. Dr. José Gilvan Rodrigues Maia (Orientador)
Universidade Federal do Ceará (UFC)

Prof. Dr. George Allan Menezes Gomes
Universidade Federal do Ceará (UFC)

Prof. Dr. Ricardo Brauner dos Santos
Universidade Federal do Ceará (UFC)

AGRADECIMENTOS

À minha família, minha mãe Vangleuma, minha prima Fran e minhas tias Girlene e, em especial, Joana D'arc, que sempre me incentivaram a buscar meus sonhos e ir além.

Aos integrantes da minha equipe, Paulo Serpa, Malu Fernandes e Hariel Tenório, que fizeram esse projeto ser possível, que estiveram comigo durante desde o segundo semestre, e que fizemos tudo juntos desde então, nos apoiando e ajudando.

Aos meus amigos Felipe, Laís, Lucas, Thiago, Isaac, Antônio, Júlia, Igor, Thays, Leonardo e Luã, que me trouxeram diversão e acolhimento nos dias mais difíceis.

A minha cadela Lina, que sempre me recebeu com enorme felicidade sempre que chegava em casa, e que fez parte dessa rede de apoio.

Aos secretários do curso de Sistemas e Mídias Digitais Monalisa Menezes e Allan George que sempre foram muito receptivos e competentes em auxiliar, instruir sobre todos os detalhes pertinentes sobre a formação acadêmica no curso, bem como também os seus atendimentos sempre com muita simpatia, humanidade e praticidade comigo e com todos os alunos do SMD.

“Os jogos não devem ser apenas divertidos. Eles devem ensinar ou despertar interesse em outras coisas.”

(Hideo Kojima)

RESUMO

Visando aproveitar o exponencial crescimento do mercado de jogos eletrônicos, a equipe Panetone desenvolveu, em 2020, *Hexachronos*, um jogo de estratégia com elementos de RPG e Visual Novel e que pudesse ser portado para plataformas mobile. Houve, contudo, falhas no processo de desenvolvimento que foram derivadas da aceleração e não realização de algumas etapas, acarretando em um produto inadequado ao contexto do público e às próprias propostas, necessitando, assim, que fosse reiniciado. Este relatório abrange portanto o esforço referente à implementação de uma nova iteração do produto, buscando novos padrões de projeto e ferramentas mais adequadas para o desenvolvimento desse jogo.

Palavras-chave: Jogos Eletrônicos. Turn-based Strategy. Visual Novel. Relato de Experiência.

ABSTRACT

In order to take advantage of the remarkable growth of the video game market, the Panetone team developed, in 2020, a strategy game with RPG and Visual Novel elements entitled Hexachronos. This game could be later ported to mobile platforms. There were, however, flaws in the development process that were derived from the acceleration and failure to carry out some steps, resulting in a product inappropriate to the public context and to the proposals themselves, thus requiring it to be restarted. This report, then, covers the part referring to the implementation of a new iteration of the project, looking for new design patterns and more suitable tools for the development of a game.

Keywords: Computer Games. Turn-based Strategy. Visual Novel. Experience Report.

LISTA DE FIGURAS

LISTA DE ABREVIATURAS E SIGLAS

Brainstorm	O brainstorming é uma dinâmica de grupo que é usada em várias empresas como uma técnica para resolver problemas específicos, para desenvolver novas ideias ou projetos, para juntar informação e para estimular o pensamento criativo.
Framework	Compilado de códigos de programação utilizado para um determinado fim dentro de alguma linguagem ou ferramenta de programação.
GDD	*Game Design Document*. Especificação de um jogo por via documental.
TBS	*Turn-based Strategy*
VN	*Visual Novel*

SUMÁRIO

1 INTRODUÇÃO

Um jogo eletrônico é um projeto de software, apesar de ser uma integração de diferentes mídias em um produto, também requer programação. Estabelecer uma arquitetura de software adequada que defina claramente os padrões de projeto a serem utilizados é de vital importância para a evolução do processo de desenvolvimento de jogos (RABIN, 2005; MAIA; VIDAL, 2003). Assim, uma arquitetura deve descrever os componentes que realizam os requisitos do produto e como esses componentes colaboram para dar vida ao jogo, ao mesmo tempo que apresente a possibilidade de desacoplar esses componentes, favorecendo tanto a manutenção daquele jogo quanto o reuso dessas soluções em outros produtos (NYSTROM, 2014). Os defeitos no desenvolvimento do software podem ocorrer em artefatos de diferentes etapas do processo de desenvolvimento, podendo estar presentes, por exemplo, em um documento de requisitos ou no código-fonte (COMMITTEE *et al.*, 1990).

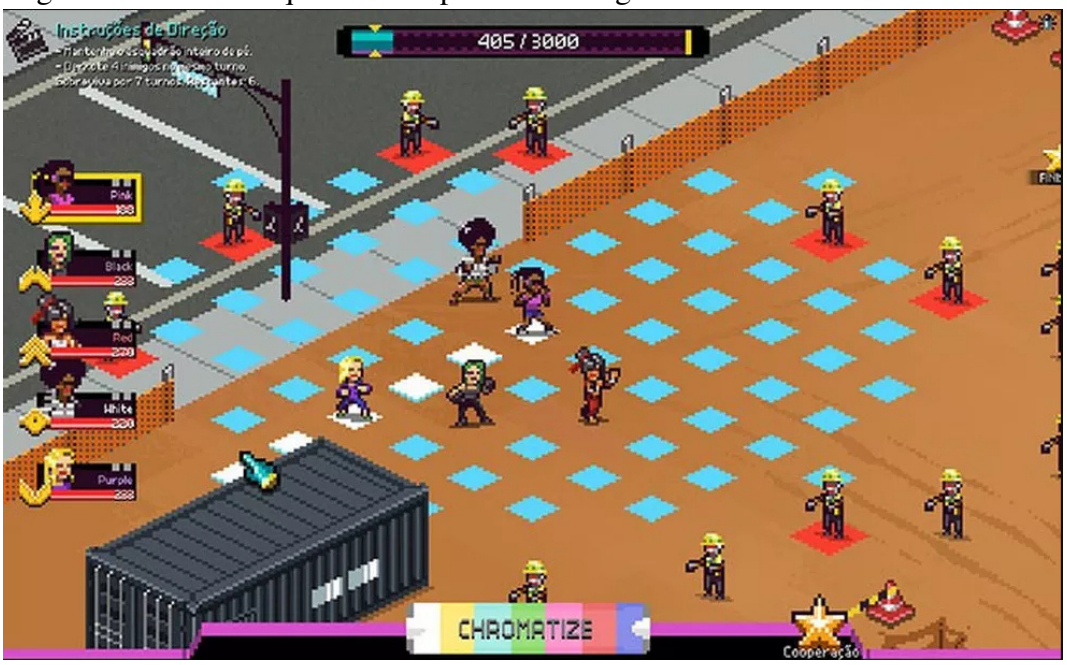
A arquitetura de software de que trata este trabalho foi idealizada no contexto da criação do jogo Hexachronos pela equipe Panetone como um entregável da disciplina de Projeto Integrado II do curso Sistemas e Mídias Digitais (SMD) da Universidade Federal do Ceará (UFC). Como elemento inovador demandado pela disciplina, a proposta do jogo consistiu na mistura de dois gêneros de jogos eletrônicos: *Turn-based Strategy* (TBS) e *Visual Novel* (VN). TBS ou Estratégia Baseada em Turnos é um tipo de jogo eletrônico de estratégia que, ao contrário de outros jogos de estratégia, não se desenrola em tempo real ou de forma concorrente, mas que se dá como uma sucessão de turnos nos quais cada jogador dispõe de tempo para elaborar as suas ações. VN ou Romance Visual, por sua vez, apresenta um estilo de jogo focado em leitura de texto e escolhas, i.e., possuem enfoque no enredo, nos quais o jogador acompanha uma história por meio de textos, músicas, imagens e vídeos.

A seguir, temos alguns exemplos de jogos eletrônicos que pertencem à esses gêneros. Na figura 1 temos Steins;Gate como exemplo de Visual Novel, na figura 2 temos Chroma Squad, como exemplo de jogo de estratégia baseada em turnos, na figura 3 temos Persona 5, que mistura os gêneros de Visual Novel e combate em turnos.

Figura 1 – Steins;Gate - Exemplo de Visual Novel

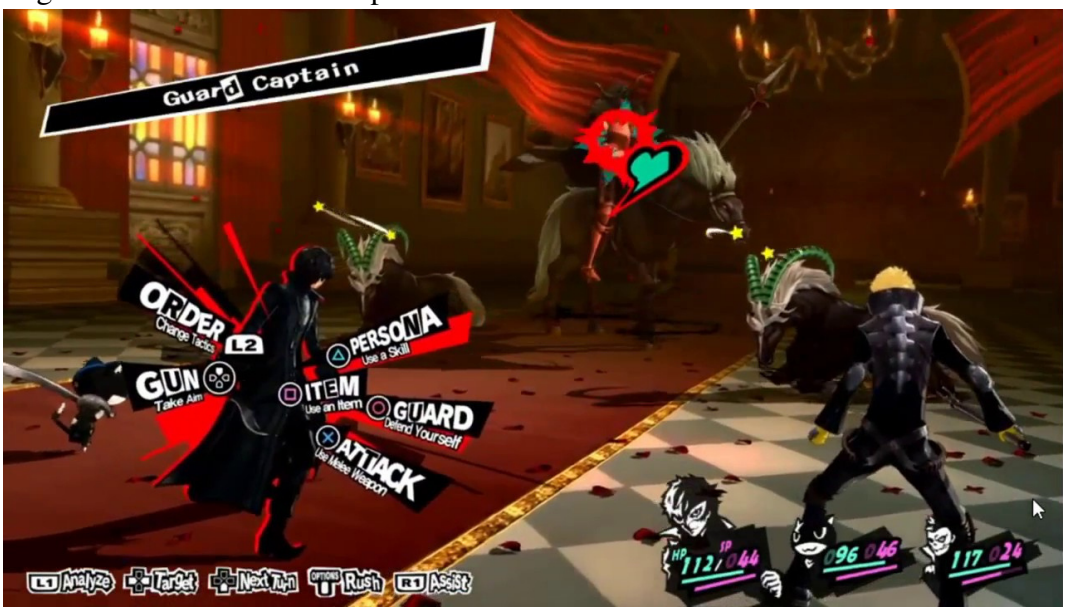
Fonte: https://www.gameblast.com.br/

Figura 2 – Chroma Squad - Exemplo de Estratégia Baseada em Turnos



Fonte: https://www.gematsu.com/

Figura 3 – Persona 5 - Exemplo de mescla entre Visual Novel e Combate em Turnos



Fonte: https://www.gematsu.com/

Dado o caráter de inovação do jogo, a equipe enfrentou diversas adversidades no seu desenvolvimento, principalmente relativas à escalabilidade do projeto, à manutenção do código e à otimização de atividades recorrentes no processo de desenvolvimento do protótipo do jogo, como, por exemplo, adição de conteúdo e de novas mecânicas. A equipe resolveu fazer uma nova iteração do Hexachronos tendo esse protótipo como base e visando melhorar vários aspectos problemáticos referentes à sua estrutura de software. A maioria dos problemas encontrados na primeira iteração do produto estão interligados em algum nível. Os principais problemas identificados poderiam ter sido contornados se houvesse maior atenção à elaboração e à aplicação de uma arquitetura de software adequada às peculiaridades do Hexachronos.

É importante destacar que, apesar do caso analisado neste trabalho ser um jogo eletrônico, a problemática de arquiteturas de software não é exclusiva do desenvolvimento de jogos, mas do desenvolvimento de softwares em geral (NYSTROM, 2014). Note-se, ainda, que os jogos eletrônicos são artefatos complexos de software que possuem vários aspectos e elementos que podem caracterizar o seu desenvolvimento, ou seja, essa é uma atividade que demanda processos que compreendam principalmente o design e a avaliação do produto (SCHELL, 2008), além das linguagens de programação e do reuso de Framework especializados (MAIA; VIDAL, 2003; MAIA, 2005).

1.1 Objetivos

1.1.1 Objetivo Geral

Relatar o desenvolvimento da segunda iteração do jogo Hexachronos, adotando uma nova e mais adequada arquitetura de software.

1.1.2 Objetivos Específicos

Os seguintes objetivos específicos foram elencados para se atingir o objetivo geral deste trabalho:

- Analisar os principais requisitos técnicos do jogo; Hexachronos;
- Determinar padrões de projeto adequados à implementação do produto;
- Conceber uma arquitetura de software para o produto; e
- Aplicar essa arquitetura no desenvolvimento do produto.

1.2 Organização do Relatório Técnico

O restante deste trabalho está organizado da seguinte maneira:

- O Capítulo 2 versa sobre a concepção inicial e definição do produto, assim como um breve relato da sua primeira iteração;
- Por sua vez, o Capítulo 3 descreve a metodologia empregada no desenvolvimento desta segunda iteração do produto, destacando aspectos como os padrões de projeto e a arquitetura adotada;
- Os detalhes de implementação são o assunto do Capítulo 4;
- O Capítulo 5 é voltado à apresentação e discussão dos resultados obtidos; e
- Por fim, o Capítulo 6 apresenta as considerações finais sobre o presente trabalho e trabalhos futuros.

2 DEFINIÇÃO DO PRODUTO

2.1 Concepção da primeira iteração

2.1.1 Design Thinking

A metodologia *Design Thinking* foi concebida por Nigel Cross (CROSS, 2011) após analisar uma pesquisa feita por Bryan Lawson sobre uma situação onde foi proposto um problema para ser resolvido por engenheiros e arquitetos, chegando à conclusão que, de uma maneira geral, cientistas e designers resolvem problemas de maneiras diferentes. A proposta inicial de *Design Thinking* tem evoluído (MELO; ABELHEIRA, 2015) sobremaneira nos últimos anos, e autores acabaram propondo alterações para seu uso em áreas bem específicas, como criação de produtos, estabelecimento de negócios, educação, publicidade, etc. A concepção primeira iteração do projeto foi feita seguindo as seguintes etapas de *Design Thinking*: *Imersão*, *Análise e Síntese*, *Ideação* e *Prototipagem*.

Figura 4 – Design Thinking.

Fonte: Autor

A imersão tem como objetivo o entendimento inicial dos problemas envolvidos no desenvolvimento do produto, com a respectiva identificação de necessidades e oportunidades que nortearão a geração de soluções. Isso requer a realização de reuniões de alinhamento da equipe de desenvolvimento com os possíveis clientes. Voltadas ao aprofundamento do contexto do problema, tais reuniões consistem na pesquisa e na discussão de assuntos análogos ao problema abordado, visando elencar soluções iniciais que sejam próximas à ideação e estabelecendo também os possíveis perfis dos usuários pretendidos. Após isso, temos a realização de entrevistas (ou qualquer outra técnica de consulta, como sessões generativas, cadernos de sensibilização, etc) com usuários dos perfis identificados, a fim de se compreender melhor seus respectivos anseios, necessidades e valores.

Ao final da imersão, a massa de dados levantados é analisada, cruzando informações a fim de identificar padrões e oportunidades. Os resultados são sintetizados de forma estruturada, empregando ferramentas de organização e planejamento comuns, porém também incluindo documentos geralmente visuais e interativos, como, por exemplo, quadros de tarefas para planejamento e organização dos esforços previstos e mapas conceituais para avaliação das relações entre produtos, tecnologias, indivíduos, etc.

A partir dos documentos gerados pela Análise e Síntese, a ideação procede por meio de Brainstorm da equipe de desenvolvimento junto a indivíduos com perfil próximo ao definido (usuários e profissionais de áreas que sejam convenientes ao tema trabalhado) e também possíveis clientes. Esses encontros para geração e debates de ideias serão os reais responsáveis pela riqueza e assertividade dos resultados pretendidos e devem, então, ser realizados com especial atenção.

Por fim, temos a fase de prototipação, que é quando o conceito inicial se torna um conteúdo formal. A fase de prototipação contempla, em um primeiro momento, a seleção e o refino de ideias, tornando-as tangíveis através de protótipos que podem ser de baixa, média ou alta fidelidade.

2.1.2 Relato da Execução do Design Thinking

A equipe foi formada por quatro membros:

- **Hariel Tenório**. Roteirista e animador;
- **Luís Eduardo**. Programador e *game designer*;
- **Paulo Vasconcelos**. Designer UI/UX ; e
- **Maria Luiza**. Ilustradora, animadora e chacter design.

Por recomendação dos professores, o método Design Thinking foi adotado. Foram feitas reuniões iniciais para definição do tipo de jogo a ser desenvolvido. Deu-se, então, início ao processo de imersão defendido pela *Design Thinking*. Foram pesquisados vários jogos, discutindo-se possibilidades que tentavam relacionar mecânicas de diferentes gêneros de jogos. Para a pesquisa de público-alvo, passou-se a trabalhar também com alunos do curso de Sistemas e Mídias Digitais da própria universidade, além de membros de grupos virtuais de desenvolvedores de jogos.

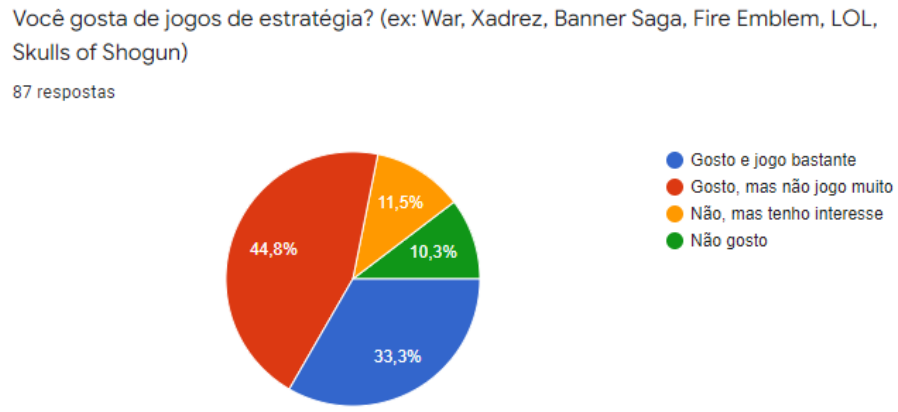
A partir dos resultados das pesquisas, definiram-se gêneros de jogos que eram populares com nosso público-alvo: Estratégia Baseada em turnos e Romance Visual. Essa pesquisa contou com a participação de 98 jovens adultos entre 20 e 39 anos, dentre esses

participantes de grupos de desenvolvimento de jogos e alunos do curso de Sistemas e Mídias Digitais, em que 57,5% dos participantes responderam que o fator mais importante de um jogo era a *gameplay*, seguida pela história do mundo e a história dos personagens. A partir disso, começamos a elaboração do GDD, documento de projeto de jogo, bastante comum na definição e posterior desenvolvimento de jogos eletrônicos, como um artefato resultante desse esforço empreendido pela equipe.

Figura 5 – Resultado da pesquisa - faixa etária.

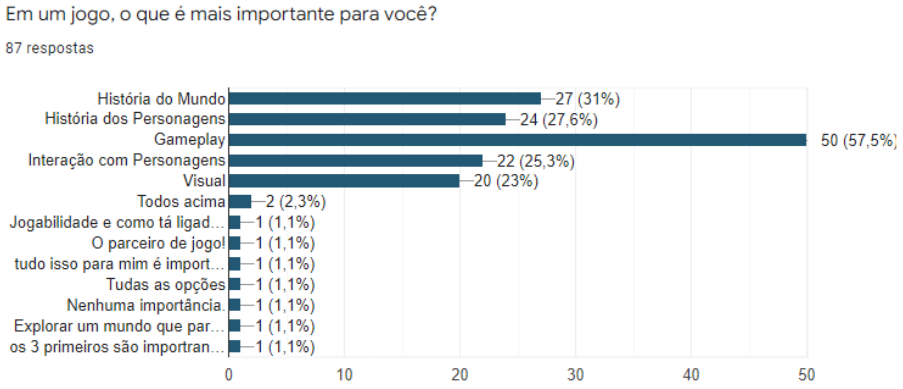
Fonte: Autor

Figura 6 – Resultado da pesquisa - jogo de estratégia.



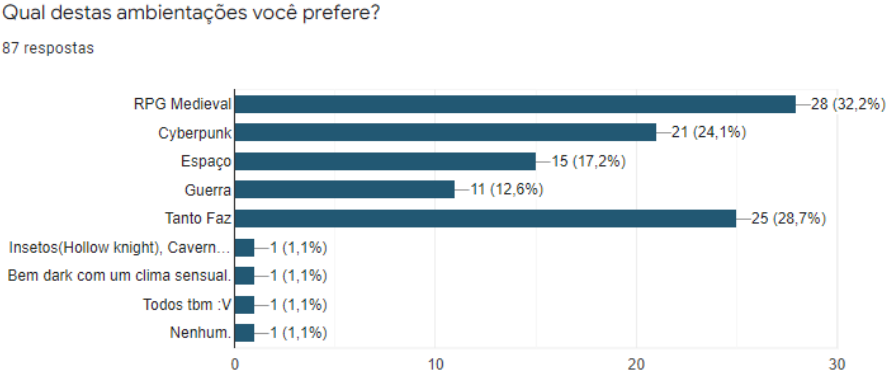
Fonte: Autor

Figura 7 – Resultado da pesquisa - preferências.



Fonte: Autor

Figura 8 – Resultado da pesquisa - ambientação.



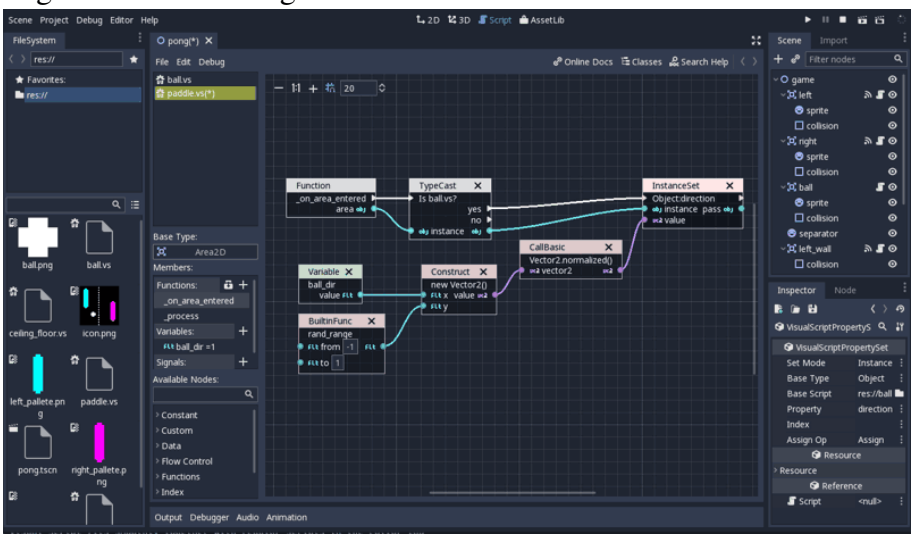
Fonte: Autor

No GDD, definimos a Tétrade Elementar (SCHELL, 2008), composta por *mecânicas*, *narrativa*, *estética* e *tecnologia*, da seguinte maneira:

- Para as mecânicas, o jogo alterna entre o modo de história, seguindo o modelo de Romance Visual, o modo de combate, seguindo o modelo de estratégia em turnos. Para o modo de combate, se seguiu a abordagem de RPG tático, onde cada personagem tem seu turno de movimento, onde pode mover-se por um espaço delimitado no jogo, e um turno de ações, onde o personagem pode realizar qualquer outra ação como atacar, defender, usar uma habilidade ou item, varia de de acordo com o escopo do jogo;
- Na narrativa, escolhemos a abordagem que modifica a narrativa de acordo com as escolhas do jogador, criando ramificações da história do jogo;
- Para a estética, nos baseamos nos resultados da pesquisa e optamos pela estética cartoon e cyberpunk; e
- Por fim, a *tecnologia* escolhida foi a *engine* Godot[1], pela familiaridade da equipe com a ferramenta.

[1] https://godotengine.org/

Figura 9 – Godot Engine - Interface.



Fonte: docs.godotengine.org/

Passou-se, então, a buscar a especificação do jogo a ser desenvolvido. Para tanto, considerou-se inicialmente a ideia de gameplay, que segundo nossa pesquisa, para nosso público-alvo, era o principal fator de um jogo. Baseado em jogos que possuem características semelhantes as apontadas pela nossa pesquisa, começamos a etapa de ideação do jogo. Dessa forma, em *Hexachronos* o jogador precisa definir estratégias baseadas nos personagens e suas respectivas habilidades para vencer batalhas em um futuro distópico no estilo cyberpunk. Além disso, também é necessária a tomada de algumas decisões ao longo da narrativa, que é contada por meio de uma *visual novel* na qual o jogador interage com outros personagens.

Por fim, chegamos a etapa de prototipagem, onde a partir das definições do jogo, foi elaborado um roteiro, anexado ao GDD. À medida que mais reuniões eram realizadas, foram estabelecidas diferentes atividades para os membros do grupo, sempre utilizando o quadro de tarefas virtual. O processo de implementação do jogo foi dividido nas seguintes etapas e respectivos protótipos:

- Definição da arte conceitual — desenhos iniciais com personagens, objetos e cenários;
- Geração de elementos gráficos — arquivos digitais contendo imagens estáticas e animadas, refinadas a partir da arte conceitual proposta;
- Definição da interface gráfica — protótipos de baixa (em papel), média e alta fidelidades (arquivos digitais) contendo elementos, devidamente posicionados em telas, necessários ao uso e correto entendimento do jogo; e
- Pesquisa de soluções de programação — porções de código responsáveis pela resolução de problemas identificados no GDD.

As diversas versões do jogo foram avaliadas pela própria equipe de desenvolvimento e por convidados que atendiam ao perfil determinado pela etapa de pesquisa.

2.2 Problemáticas no Desenvolvimento do Jogo

Problemáticas relacionadas ao desenvolvimento de jogos podem surgir da concepção inicial da arquitetura de software. À medida em que o projeto avança, essas problemáticas podem se tornar cada vez mais difíceis de se contornar, requerendo uma reestruturação do projeto como um todo. Nossa equipe encontrou algumas dessas durante a concepção da primeira iteração do projeto. A escalabilidade de um projeto como um jogo pode ser um problema se o planejamento não for bem realizado. Na medida em que uma equipe de desenvolvimento de software se conscientiza dos defeitos de alta ocorrência, tais defeitos tendem a ser evitados no momento do desenvolvimento dos próximos artefatos, economizando tempo que seria gasto em manutenções corretivas posteriores (FAGAN, 2002).

O processo iterativo e contínuo de desenvolvimento pode gerar trechos de código desnecessários, que gastam tempo de processamento da máquina, por isso, é importante ter uma arquitetura que facilite a otimização de processos, deixando o sistema mais leve, compacto e robusto. Adicionalmente, as organizações devem fazer do aprendizado com seus erros um instrumento para aumentar sua competitividade e qualidade (KALINOWSKI; TRAVASSOS, 2012). Assim, para garantir a qualidade do processo de inspeção, é de suma importância que os conceitos e técnicas envolvidos nas atividades relacionadas à inspeção estejam bem assimilados e sejam aplicados devidamente.

Conforme os jogos ao longo dos anos se tornaram sistemas grandes e complexos, a indústria de videogames está enfrentando vários desafios de engenharia de software. Kanode e Haddad (KANODE; HADDAD, 2009) identificaram os desafios da engenharia de software no desenvolvimento de jogos como:

- **Assets Diversos**. O desenvolvimento de jogos não é simplesmente um processo de produção de código-fonte, mas envolve ativos como modos 3D, texturas, animações, som, música, diálogo, vídeos;
- **Escopo do Projeto**. A escala de um videogame pode ser enorme e a indústria de jogos é conhecida por descrever e definir mal o escopo do projeto;
- **Publicação de Jogos**. Trazer um videogame ao mercado envolve convencer um editor de jogos a fazer backup financeiramente afetar as entregas e o processo de desenvolvimento;

- **Gerenciamento de Projetos**. O gerenciamento de projetos no desenvolvimento de jogos pode ser ainda mais difícil devido a cronogramas muito apertados e envolvimento de muitas profissões;
- **Organização da Equipe**. Envolve a construção de equipes que melhoram a comunicação entre as disciplinas;
- **Processo de Desenvolvimento**. Inclui mais do que apenas desenvolvimento de software; e
- **Tecnologia de Terceiros**. Para muitos desenvolvedores de jogos, o software de terceiros representa o núcleo do projeto devido aos custos crescentes de desenvolvimento e complexidade crescente.

Este relatório técnico terá enfoque no gerenciamento de *Assets* diversos, no escopo do projeto e no processo de desenvolvimento.

3 METODOLOGIA

A metodologia do trabalho é baseada em uma pesquisa sobre padrões de projeto, que se constitui de levantamentos bibliográficos como artigos e dissertações de outros autores que realizaram trabalhos relacionados ao tema anteriormente. Pesquisa esta que foi feita através da leitura de artigos, livros e monografias. Portais como Google Acadêmico, *ResearchGate* e o SciELO (*Scientific Electronic Library Online*) também serviram como ferramentas de pesquisa durante este trabalho.

3.1 Padrões de Projeto

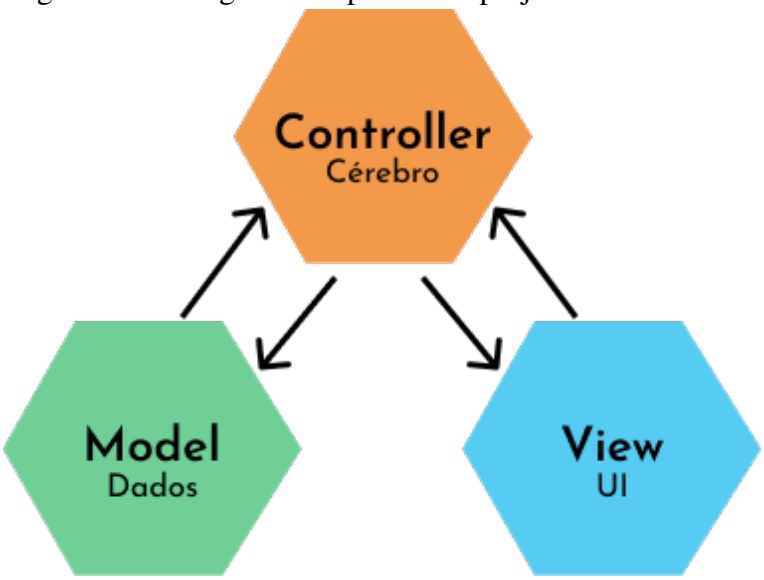
Padrões de projeto são um mecanismo para expressar a experiência de um projeto orientado a objetos. Padrões de projeto identificam, nomeiam e abstraem temas comuns em um projeto orientado a objetos. Eles capturam a intenção por trás de um projeto identificando objetos, suas colaborações e a distribuição de responsabilidades. Os padrões de projeto desempenham muitos papéis no processo de desenvolvimento: eles fornecem um vocabulário comum para o projeto, reduzem a complexidade do sistema nomeando e definindo abstrações, constituem uma base de experiência para construir software reutilizável e atuam como blocos de construção a partir dos quais projetos mais complexos podem ser construídos. Padrões de projeto podem ser considerados microarquiteturas reutilizáveis que contribuem para uma arquitetura geral do sistema (GAMMA *et al.*, 1993).

Ao discutir diferentes arquiteturas (RABIN, 2005), muitas vezes falaremos sobre acoplamento. O acoplamento é uma medida de quão firmemente duas partes do código estão conectadas uma à outra. Um acoplamento fraco significa que há apenas uma ligeira conexão entre os dois conjuntos de código, que é a situação ideal porque permite mudar uma sem afetar o outro. Quanto maior o acoplamento, mais difícil é modificar ou substituir um sem afetando o outro. Acoplamento realmente forte significa que os dois conjuntos de código são altamente dependentes um do outro.

Veremos a seguir o padrão adotado na primeira versão do projeto, e os padrões adotados na sua segunda versão.

3.1.1 MVC

Figura 10 – Diagrama do padrão de projeto MVC.



Fonte: Adaptado de freecodecamp.org (2022)

Na primeira iteração do projeto, o padrão de projeto escolhido foi o MVC (Model - View - Controller, vide Figura 10). Esse modelo (BARROS *et al.*, 2007) determina a separação de uma aplicação em três elementos:

- **Model**. É formado por entidades que representam os dados da aplicação;
- **View**. Tem por objetivo realizar a apresentação destes dados e capturar os eventos do usuário; sendo representada pelas telas; e
- **Controller**. Faz a ligação entre o Model e a View, realizando o tratamento dos eventos, atuando sobre o Model e alterando os elementos da View para representar a nova forma dos dados.

A camada de Model era composta por entidades base que são instanciadas na Controller (BasicCharacter, Buff, etc), a camada de View era composta das cenas (arquivos .tscn, nativos da ferramenta Godot) e a camada de Controller era composta de arquivos de código (.gd, também nativos da ferramenta Godot), cada um ligado à sua respectiva cena da View. Dessa forma, a camada Controller ficou responsável tanto pela exibição de elementos gráficos quanto da lógica do jogo, o que dificultou o escalonamento e manutenção do projeto. Para evitar que o problema se repetisse na próxima iteração, optamos por escolher outro padrão de projeto.

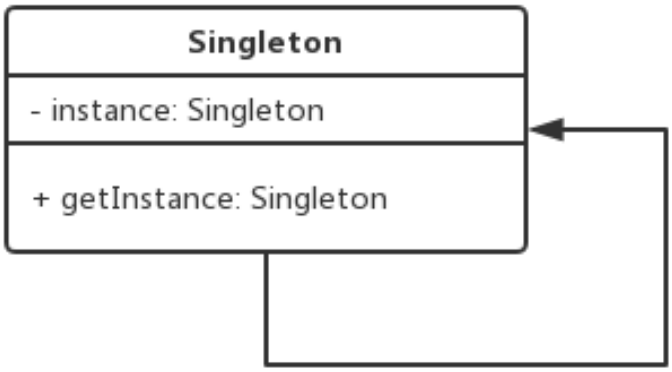
3.1.2 Singleton

Singleton é um padrão de projeto de software, do tipo criacional – isso significa que esse padrão lida com a instanciação de objetos (NYSTROM, 2014; GAMMA *et al.*, 1993).

Assim, apesar de polêmicas com relação ao seu uso (NYSTROM, 2014), o padrão Singleton garante a existência de apenas uma instância de uma classe, mantendo um ponto global de acesso ao seu objeto.

Singletons podem ser uma ferramenta útil ao desenvolver com Unity graças à forma como as classes são instanciadas. Esse método é preferível à ocultação do construtor, pois é possível instanciar um objeto com um construtor oculto no Unity. Para evitar que uma instância seja substituída, uma verificação deve ser executada para garantir que a instância seja nula. Se não for nulo, o GameObject que contém o script incorreto deve ser destruído.

Figura 11 – Diagrama do padrão de projeto Singleton.



Fonte: programmerlib.com

Se outros componentes forem dependentes do singleton, a ordem de execução do script deve ser modificada para garantir que o componente que define o singleton seja executado primeiro.

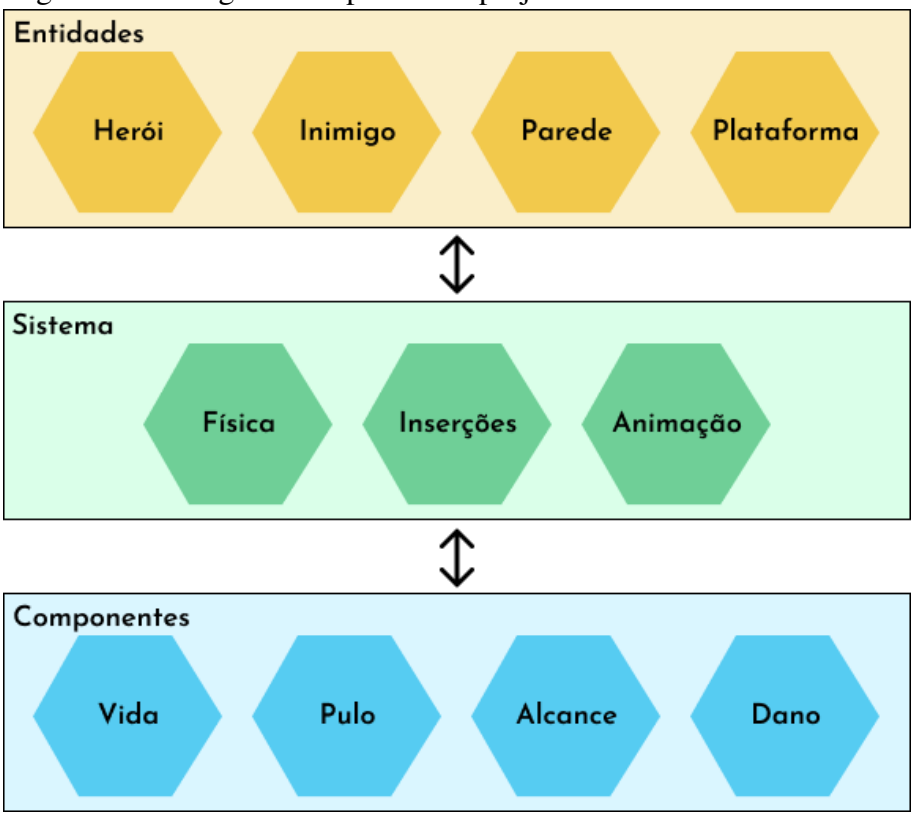
No entanto, há algumas considerações a serem feitas sobre as consequencias do uso do Singleton. De acordo com Rabin (RABIN, 2005), Singletons nos dão duas coisas principais: uma única instância e acessibilidade global. Como é tão diferente de uma variável global? Não é muito diferente. Na verdade, o único benefício sobre uma variável global é que mantemos mais controle sobre como ela é criada e destruída. No que diz respeito a todo o resto, pode ser considerada uma variável global. O que sabemos sobre variáveis globais? Eles não levam a um código modular, e podem ser um problema sério em grandes bases de código ao entendê-los e mantê-los. Os mesmos avisos sobre variáveis globais se aplicam a singletons, portanto, não os use a menos que você precise.

3.1.3 ECS

Segundo Wiebusch (WIEBUSCH; LATOSCHIK, 2015), o padrão Entity-Component-System (ECS) se tornou o principal padrão de projeto usado em arquiteturas modernas para estruturas de um Sistema Interativo em Tempo Real (RIS, *Real-time Interactive System*).

O padrão desacopla diferentes aspectos de uma simulação, como gráficos, física ou IA verticalmente. Seu objetivo principal é separar algoritmos, fornecidos por módulos de simulação personalizados de alto nível ou motores, da estrutura de objetos das entidades de baixo nível simuladas por esses motores. Nesse contexto, ele mantém as vantagens da programação orientada a objetos, como encapsulamento e controle de acesso.

Figura 12 – Diagrama do padrão de projeto ECS.



Fonte: Adaptado de researchgate.net (2022)

O ECS tem três camadas principais:

- **Entidades**. As entidades, ou coisas, que povoam o jogo;
- **Componentes**. Os dados associados às entidades, porém organizados pelos próprios dados em vez de por entidade (Essa diferença na organização é uma das principais diferenças entre um design orientado a objetos e um orientado a dados); e
- **Sistemas**. A lógica que transforma os dados do componente de seu estado atual para seu próximo estado - por exemplo, um sistema pode atualizar as posições de todas as entidades móveis por sua velocidade vezes o intervalo de tempo desde o quadro anterior.

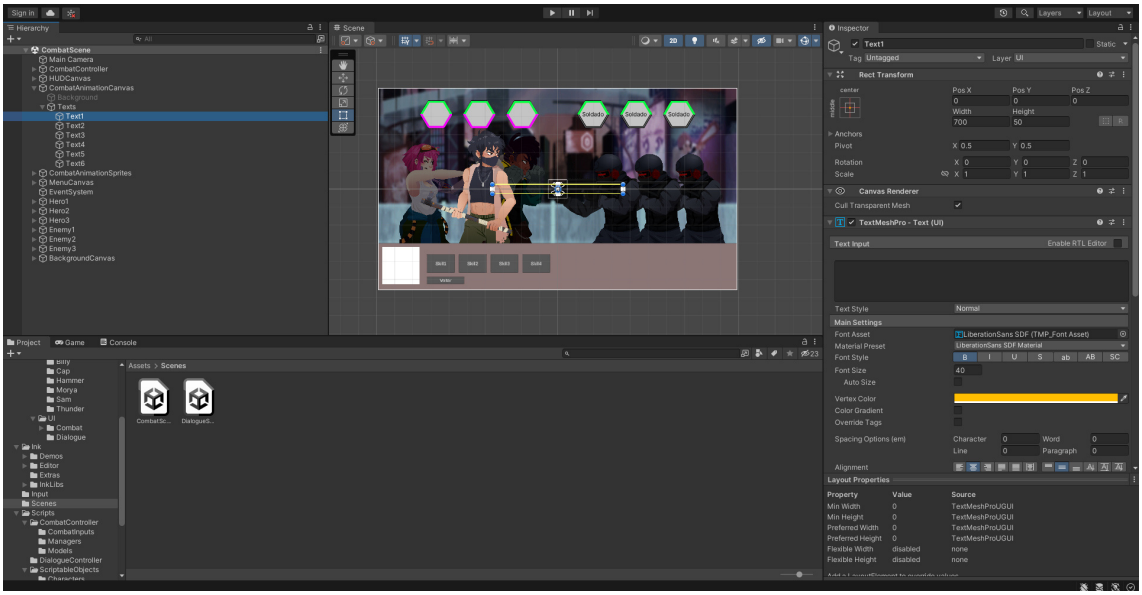
Para esta nova iteração, iremos aplicar o padrão ECS no projeto.

3.2 Arquitetura Proposta

O material encontrado sobre o padrão de projeto ECS era bastante referenciado na documentação da *engine* Unity. Optamos por trocar a *engine* da primeira iteração (Godot) para uma outra (Unity) – vide Figura 13. O padrão de projeto Singleton se mostra bastante útil na ferramenta escolhida, apesar do custo do acesso global já citado. O padrão adotado para a

estrutura do projeto sera1 o Entity-Component-System.

Figura 13 – Unity Engine - Interface.



Fonte: Autor

4 IMPLEMENTAÇÃO

Este capítulo tem por objetivo apresentar os principais aspectos de implementação do novo protótipo. São detalhadas as tomadas de decisões e as estratégias utilizadas para o desenvolvimento de cada elemento do escopo do novo projeto.

4.1 Mecânicas Principais

Como definido anteriormente, o jogo alterna entre dois modos: o modo de história, que segue o modelo de Romance Visual; e o modo de combate, que segue o modelo de Estratégia em Turnos. Para isso, foram criadas duas cenas da *Unity* — que, para efeitos práticos, são dois arquivos no formato *.unity*. Essa escolha permitiu implementarmos separadamente cada uma das mecânicas e também dividir esse esforço de forma mais racional.

Na primeira implementação do projeto (em Godot), existia um único arquivo de código-fonte (no formato *.gd*) e um arquivo de cena (no formato *.tscn*) para cada mecânica. Note-se que Godot utiliza uma linguagem de script própria, GDScript, para a programação nativa nesse motor. Além disso, existem bindings para outras linguagens, tais como C#, VisualScript e C/C++.

No projeto em Godot, os elementos visuais eram dispostos no arquivo de cena, ou inseridos programaticamente por meio de chamadas no arquivo de código. As demais regras de negócio referentes à mecânicas do jogo eram feitas nesse mesmo arquivo de código. À medida que a implementação evoluía, mais responsabilidade era atribuída a esse único arquivo, tornando cada mais difícil sua manutenção e coerência. Assim, utilizamos o padrão ECS para contornar esse problema na nova implementação.

No ECS, criamos entidades (o "E" do ECS), podendo estas serem o personagem principal, inimigos, objetos do cenário, etc. Criamos então os componentes (o "C" do ECS) que serão acoplados às entidades criadas através do sistema (o "S" do ECS). Exemplos de componentes são os comportamentos a serem assimilados pelas entidades, tais como pular, atirar, andar, etc. No nosso caso, ao invés de termos entidades na cena que receberão seus respectivos componentes, a própria cena será interpretada como uma grande entidade agregadora. Os elementos visuais são dispostos na cena e manipulados pelos diferentes componentes assimilados a ela.

Criamos *GameObjects* na cena e então atribuímos um código C# à estes objetos. Na

prática, tratam-se de especializações da classe *MonoBehaviour* da Unity 3D, sendo que cada classe possui uma função específica (e.g., pular, atacar, etc).

4.1.1 Cena de Diálogo

Na *DialogueScene* (ou cena de diálogo), será realizada a mecânica de Visual Novel. Serão exibidos: um *background*; uma imagem do personagem; uma caixa de texto com o nome e a fala do personagem; e um modo de seleção de opções para quando o jogador precisar fazer alguma escolha de diálogo, que poderá, ou não, influenciar na narrativa.

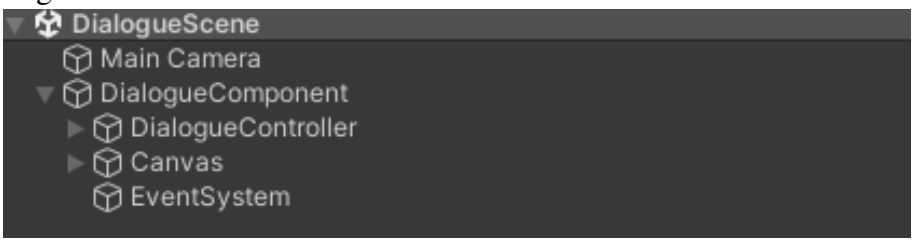
Para a manipulação de diálogos nas cenas, utilizamos uma extensão da Unity que facilita a leitura de arquivos no formato *.ink* – um formato para escrita de roteiros em games[1].

Os seguintes componentes estão presentes nessa cena:

- **InputManager**. Trabalha os inputs do usuário. Necessário para que o jogador possa passar para a próxima fala e escolher a opção desejada na parte de escolha de diálogo;
- **DialogueTrigger**. Recebe os arquivos *.ink* e manipula a sequência de diálogos enquanto gerencia as escolhas do jogador. Esse componente é o controlador principal da cena. Além disso, é o único a ser substituído em cada instância da cena de diálogo;
- **DialogueManager**. Manipula os dados recebidos pelo *DialogueTrigger* e então atualiza os componentes visuais da cena, tais como a caixa de texto de fala, caixa de texto do personagem, botões de escolha (caso sejam necessários pelo contexto), background, sons e imagens de personagens;
- **CharacterImageManager**. Componente auxiliar ao *DialogueManager*, manipula exclusivamente a foto do personagem que está falando no momento. Com isso, é possível atrelar animações ou mesmo quadros com a situação ou emoção do personagem enquanto o diálogo se desenrola; e
- **SoundManager**. Componente auxiliar ao *DialogueManager*, manipula exclusivamente os sons do personagem que está falando no momento.

[1] https://www.inklestudios.com/ink/

Figura 14 – Pastas da cena CombatScene.



Fonte: Autor

Utilizamos então o padrão *Singleton* para cada um desses componentes, exceto o *DialogueTrigger*. Dessa forma, quando o *DialogueTrigger* chamar um novo diálogo, chamará o método responsável por isso presente no *DialogueManager* e não precisará criar uma nova instância para quando precisar chamar essa função novamente. O mesmo se aplica para quando o *DialogueManager* mudar a imagem do personagem ou emitir um som, precisando apenas chamar o método presente nos seus componentes auxiliares. Dessa forma, conseguimos distribuir as funções necessárias para a realização das mecânicas sem precisar deixar tudo isso em um único arquivo de código, além de permitir reaproveitar um mesmo componente em diferentes cenas.

Figura 15 – Implementação do Singleton em um Componente.

```
public class InputManager : MonoBehaviour
{
    private static InputManager instance;
    Mensagem do Unity | 0 referências
    private void Awake()
    {
        if (instance != null)
        {
            Debug.LogError("Found more than one Input Manager in the scene.");
        }
        instance = this;
    }
    1 referência
    public static InputManager GetInstance()
    {
        return instance;
    }
}
```

Fonte: Autor

4.1.2 Cena de Combate

Na *CombatScene* (ou cena de combate), será realizada a mecânica de Combate em Turnos. São exibidos: um *background*; os personagens controlados pelo jogador; os inimigos; a barra de vida e energia de cada personagem presente na cena; e alguns menus de combate (menu principal, de escolha de habilidade e de escolha de alvo).

Os seguintes componentes estão presentes nessa cena:

- **InputManager**. Trabalha os inputs do usuário. Necessário para que o jogador possa navegar entre os menus e possa atacar, defender, escolher uma habilidade e escolher alvos;
- **CombatCharManager**. Define quais personagens estarão presentes naquele combate em questão e delega funções para os demais componentes. Esse componente é o controlador principal da cena de combate;
- **CombatUIManager**. Controla os elementos de UI relativos aos menus (principal, de escolha de habilidade e de escolha de alvo). O ciclo de turnos é controlado por esse componente;
- **SkillManager**. Responsável por trabalhar os diferentes efeitos de cada habilidade individualmente;
- **CharStatsManager**. Contém um lista de *BasicCharInfo*, um *ScriptableObject* criado para conter informações básicas de um personagem. Consultamos os dados que definem o estado dos personagens utilizando esse componente;
- **CharSpriteManager**. Componente auxiliar ao *CombatCharManager*, responsável por manipular as imagens dos personagens presentes na cena, assim como seus retratos na HUD;
- **BuffIconManager**. Auxiliar ao *CombatCharManager*, responsável por manipular as imagens dos *buffs* ou *debuffs* (aprimoramentos ou degradação de status) presentes na HUD de cada personagem;
- **SoundManager**. Componente auxiliar ao *DialogueManager*, manipula exclusivamente os sons do personagem que está falando no momento; e
- **CombatAnimationManager**. Componente auxiliar ao *CombatCharManager*, responsável por manipular o estado da animação de algum personagem quando este ataca, defende, usa uma habilidade, é atacado ou é alvo de alguma habilidade do inimigo.

Figura 16 – Pastas da cena CombatScene.

Fonte: Autor

Utilizamos então o *Singleton* para cada um desses componentes, assim como na *DialogueScene*. O componente *InputManager* utilizado nessa cena é o mesmo utilizado na cena *DialogueScene*. Dessa forma, caso futuramente precisemos colocar diálogos durante o combate, basta assimilarmos na cena os componentes relativos à mecânica em questão.

5 RESULTADOS

Este capítulo apresenta e discute os resultados da implementação da nova iteração, comparando, sempre que possível, aspectos da nova iteração e em relação à anterior.

5.1 O Jogo Implementado

As mudanças entre a primeira e segunda iteração do projeto descritas nesse relatório são referentes apenas à implementação das mecânicas principais do jogo. No momento da escrita deste documento, a segunda iteração possui alguns *placeholders* que não estarão presentes na versão final do jogo. O desenvolvimento das duas versões foi feita por um único programador.

O leitor interessado pode conferir os vídeos demonstrativos dessas mecânicas. Para tanto, basta seguir os links presentes nas respectivas notas de rodapé:

- **Primeira Iteração**
  1. Mecânicas de Combate em Turnos[1]; e
  2. Diálogos e Escolhas[2].
- **Segunda Iteração**
  1. Combate em Turnos[3]; e
  2. Diálogos e Escolhas[4].

5.2 Análise Comparativa

A seguir são desenvolvidas comparações entre as mecânicas e demais detalhes entre as duas iterações do produto.

5.2.1 Sistema de Combate

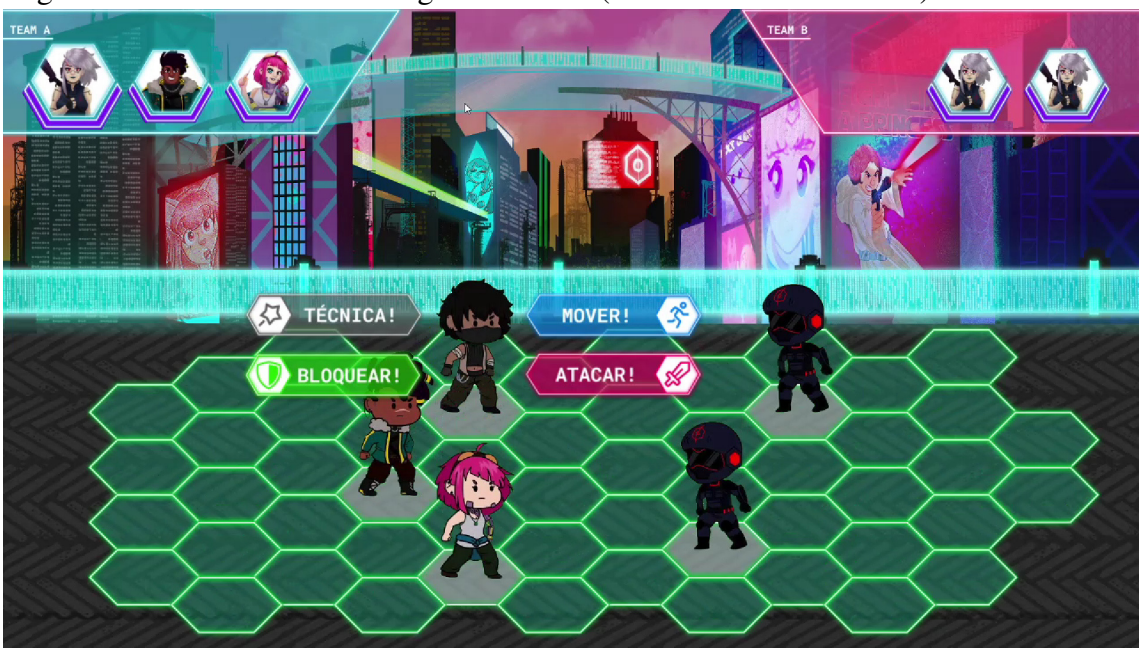
Na primeira versão do projeto, feito em Godot, o combate possuía uma camada a mais de complexidade: a parte de RPG Tático. Nessa parte, cada personagem possuía uma ação de movimento e uma ação de combate por turno. Essa ação de combate podia ser um ataque direto, o uso de uma habilidade, um bloqueio ou passar a vez. A quantidade máxima de personagens controlados pelo jogador é 3, e a quantidade máxima de personagens inimigos é 4.

[1] https://youtu.be/_cLTr2K9vFU
[2] https://youtu.be/JyZNNiVxyWk
[3] https://youtu.be/Cly9SAqD3n0
[4] https://youtu.be/0HS0TTxH3DI

A quantidade de casas (ladrilhos) em que o personagem poderia andar variava de acordo com os seus atributos e características. Assim, um algoritmo de *pathfinding* era utilizado para marcar visualmente na tela as opções possíveis de destino para o personagem. Todavia, essa implementação possuía limitações significativas. Em primeiro lugar, a responsabilidade de gerenciamento das mecânicas sendo colocada num único controlador. Além disso, foi usada uma implementação pouco otimizada do algoritmo de *pathfinding*, o qual possuía complexidade assintótica $O(n^3)$.

A performance do combate era bem pouco fluida quando os personagens utilizavam habilidades mais complexas ou quando tinham um grande alcance de locomoção. Isso atrapalhava bastante a experiência do jogador. O tempo de desenvolvimento deste sistema de combate foi de um mês e meio, resultando em um controlador principal com 476 linhas de código.

Figura 17 – Combate de estratégia em turnos (Hexachronos - 1ª versão).



Fonte: Autor

Na segunda versão, feita em Unity, foi removida a parte de RPG tático. Essa foi uma decisão de *game design* que deixou apenas uma ação de combate por turno para cada personagem. Isso resultou em um *gameplay* mais ágil, de acordo com alguns *feedbacks* dos testadores.

Além disso, as habilidades de combate agora afetam um inimigo, vários inimigos, um aliado, vários aliados ou o próprio personagem. A divisão de responsabilidades entre os diversos componentes provou-se uma escolha eficiente, visto que não houveram *bugs* perceptíveis ou perda na fluidez durante os testes iniciais do jogo.

Os algoritmos iterativos de maior ordem foram de $O(n \times m)$, sendo n o número de

personagens e *m* o número de *buffs* ativos em cada personagem. É preciso destacar que há apenas um máximo de 3 personagens em cada um dos dois lados envolvidos do combate e que o número máximo de *buffs* ativos por personagem é 4.

O tempo de desenvolvimento desse sistema de combate foi de um mês e meio. O controlador principal resultante ficou com 410 linhas de código. São 2.181 linhas de código na soma de todos os componentes acoplados à entidade. É importante ressaltar que nessa versão, cada um dos 8 personagens jogáveis possui 4 habilidades, cada qual com efeitos diferentes.

Note-se que cada habilidade é tratada no componente auxiliar *SkillManager*, que é o mais extenso componente do projeto. Nessas habilidades, temos um gasto de energia diferente para cada uma, além da possibilidade de aplicação de efeitos temporários (o tempo sendo relativo ao número de turnos) positivos e negativos nos personagens. As características temporariamente alteradas pelos efeitos são:

- Ataque;
- Defesa;
- Chance de acerto;
- Chance de esquiva;
- Multiplicador de dano crítico; e
- Chance de acerto crítico;

Além desses efeitos, há alguns *status* negativos que aplicam penalidades para o personagem que os possuir. São estes os status negativos aplicáveis até o momento:

- Atordoado. Impedindo o personagem de realizar qualquer ação;
- Sangramento. Causando um pouco de dano toda vez que chegar a vez do personagem agir; e
- Provocar. Força os oponentes controlados por IA à atacar o personagem;

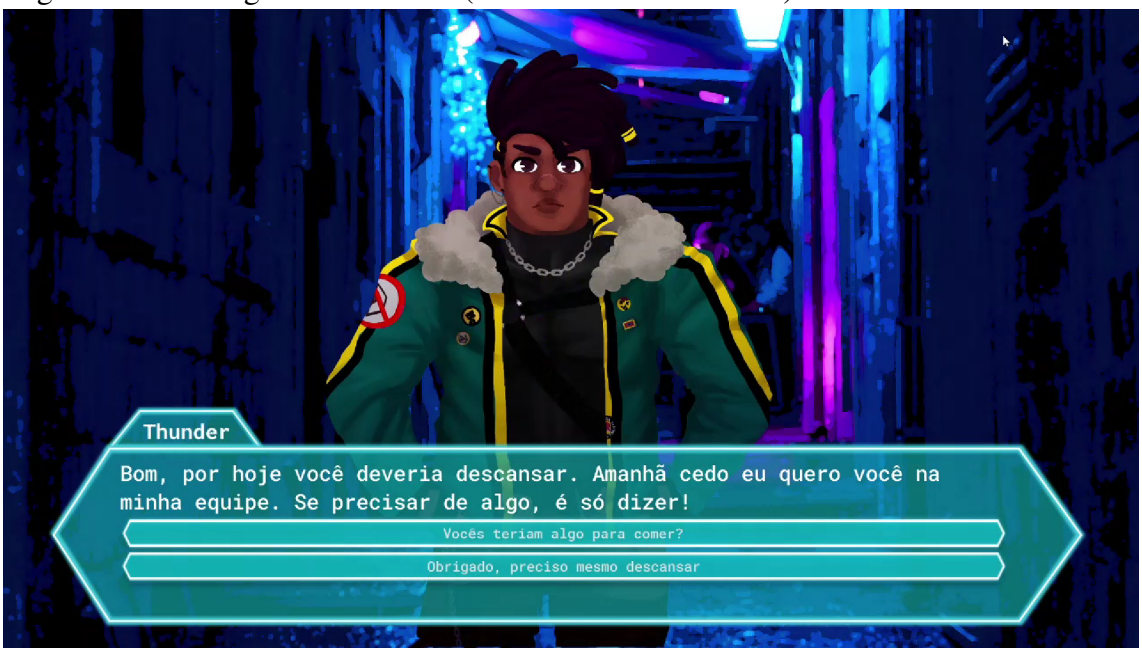
Figura 18 – Combate de estratégia em turnos (Hexachronos - 2ª versão)

Fonte: Autor

5.2.2 Sistema de Diálogo

Para a parte de diálogo, feita em Godot, um único arquivo .JSON continha todas as falas e ramificações de diálogo do jogo. Uma consulta era feita, sem o uso de nenhum framework dedicado à tarefa, e os diálogos eram dispostos na tela através de um roteiro feito via código, que ordenava manualmente as falas e respostas. Por fazer sempre essa chamada, o carregamento de assets acabava ficando cada vez mais demorado, à medida que assets novos entravam na cena. No entregável da primeira versão, cada cena demorava alguns segundos para carregar todos os assets. O tempo de desenvolvimento deste sistema de combate foi de um mês e meio, e o controlador principal ficou com 563 linhas de código.

Figura 19 – Diálogo com escolhas (Hexachronos - 1ª versão)

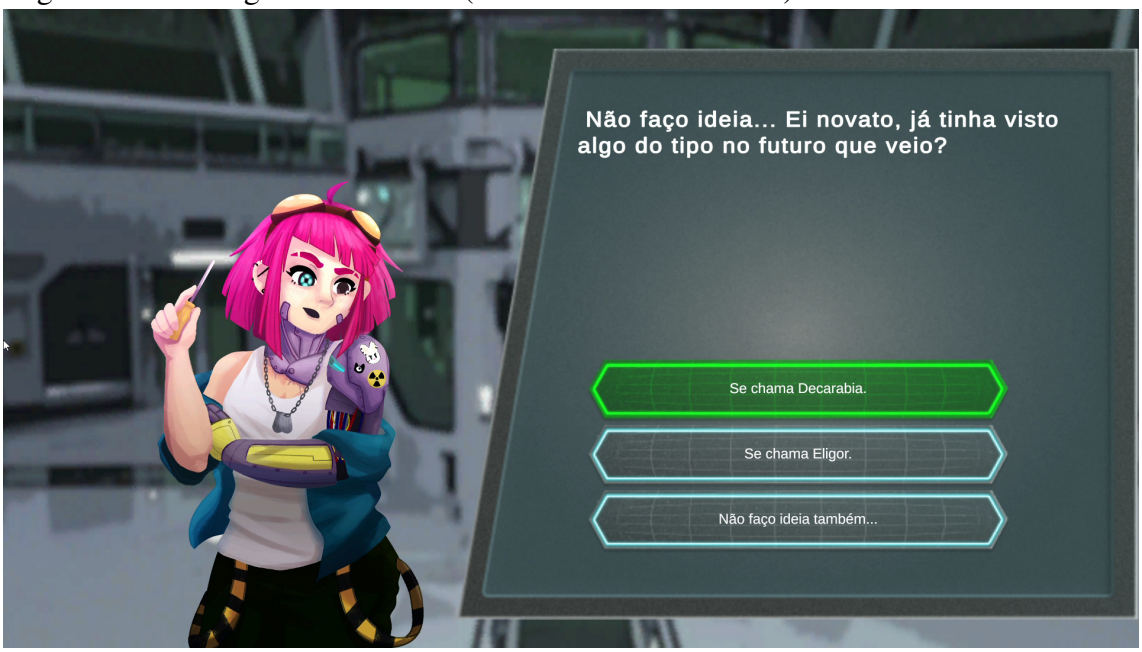


Fonte: Autor

Na segunda versão, desenvolvida em Unity, foi utilizado um *framework* feito especialmente para o gerenciamento de diálogos, *Inky*. Este *framework* utiliza arquivos no formato *.ink* e que podem ser utilizados com ferramentas de escrita de roteiro, mesmo sem ser de videogames. O carregamento de assets faz a chamada apenas do assets necessário para o contexto do momento, evitando uma alocação desnecessária de memória para esses arquivos que muitas vezes nem são utilizados na cena.

A lógica por trás das escolhas e as ramificações de roteiro atreladas à elas era responsabilidade do próprio framework Inky, deixando o código mais limpo e mais fácil de realizar manutenção. O tempo de desenvolvimento deste sistema de combate foi de um mês e meio, e o controlador principal ficou com 40 linhas de código. A soma das linhas de código com todos os demais componentes é 251.

Figura 20 – Diálogo com escolhas (Hexachronos - 2ª versão)



Fonte: Autor

5.2.3 Comparativo entre as duas versões

Apesar da segunda versão ter ficado com mais linhas de código (2.882 linhas no total) do que a primeira versão (1.039 linhas no total), sua estrutura permitiu que ficasse mais prático a adição de código e sua manutenção, além de permitir que alguns componentes pudessem ser utilizados em mais de um contexto, como é o caso do *InputManager*. Essa estrutura permite inclusive que, futuramente, possamos colocar o sistema de diálogos embutido no sistema de combate sem grande esforço.

A maneira como a *engine* gerencia os diversos *assets* sem prejudicar o desempenho do jogo também é outra melhoria a ser mencionada aqui. Ter uma estrutura mais robusta possibilitou aumentar a complexidade do jogo. Na primeira versão, cada personagem poderia se mover, e fazer uma ação de ataque, defesa, ou usar uma habilidade única. Cada personagem possuía uma habilidade apenas, sendo o efeito dessas habilidades um dano ou cura, podendo ser em área ou em um alvo específico.

Na segunda versão, apesar da capacidade de mover-se pela cena de batalha ter sido removida, cada personagem possui quatro habilidades únicas, que podem causar dano e/ou aplicar efeitos temporários positivos ou negativos, podendo afetar o próprio personagem, algum aliado, todos os aliados, algum inimigo ou todos os inimigos. Esse sistema de efeitos foi adicionado nessa segunda versão. A manipulação de ScriptableObjects, contendo informações básicas tanto dos personagens quando de suas habilidades, possibilita que novos personagens

e habilidades sejam criadas futuramente e acopladas ao jogo sem precisar sequer modificar ou adicionar novas linhas de código.

6 CONSIDERAÇÕES FINAIS

Embasada pelo significativo crescimento da indústria de jogos nos últimos anos e sua popularização em dispositivos móveis, A equipe Panetone iniciou o desenvolvimento de um jogo digital de estratégia em turnos que continha elementos de RPG e Visual Novel que pudesse ser portado para a plataforma *mobile*. Durante o desenvolvimento desse jogo, entretanto, ocorreram falhas que resultaram em um projeto inconsistente e não adaptado ao contexto do usuário. Ao perceber essas falhas, a equipe, então, decidiu reiniciar o projeto para corrigi-las.

Dessa forma, este relatório técnico-científico buscou apresentar uma proposta de uma nova implementação das principais mecânicas do jogo para que essa se adéque às mudanças de mecânica, de arte e, principalmente, ao contexto e necessidade dos jogadores. O objetivo principal deste relatório foi relatar o desenvolvimento da segunda iteração do jogo *Hexachronos*, adotando uma nova e mais adequada arquitetura de software.

Para isso, foram utilizados a *engine* Unity e o padrão de projeto ECS (*Entity-Component-System*), elaborado para o desenvolvimento de jogos. Essas escolhas foram aliadas às habilidades aprendidas ao longo do curso SMD (e.g., técnicas de otimização de código e uso de padrões auxiliares como o *Singleton*).

Através da descrição de implementação, é possível perceber como a estrutura do código e do projeto como um todo mudaram bastante. O novo padrão ECS facilitou entender o que cada parte fazia e como essas partes se comunicavam, passando e recebendo informações entre elas. Além disso, o acoplamento de novas melhorias ou mecânicas também foi simplificado.

No começo do desenvolvimento da nova versão do projeto, foram analisados os principais requisitos técnicos do jogo. Com a mudança da mecânica de combate, agora sendo apenas por turno, não tendo mais a parte de movimentação tática, alguns empecilhos como o algoritmo de *pathfind* com complexidade assintótica $O(n^3)$ foram removidos. O foco foi desenvolver as duas principais mecânicas do jogo: Diálogos com Escolhas e Combate em Turno. Como requisito não-funcional do jogo, essas duas mecânicas devem funcionar sem problemas de performance para o usuário, com carregamento rápido de *assets* e pouca perda de *frames* durante sua execução.

Em seguida, os padrões ECS e Singleton substituíram o antigo padrão MVC, que demonstrou alguns problemas na implementação de um jogo. Então essa estrutura foi aplicada ao projeto em *Unity*, produzindo um resultado satisfatório ao comparar com o resultado da antiga versão. A evolução foi percebida tanto em performance quanto em capacidade de realizar

manutenção e melhorias. Dessa forma, o objetivo final da revisitação do projeto foi cumprido, assim com cada um dos objetivos específicos definidos neste TCC.

Por fim, estabelecem-se, como pretensões para trabalhos futuros: (1) concatenar as cenas que utilizam essas mecânicas numa sequência lógica para a narrativa criada; (2) realizar contínuas melhorias no código; (3) desenvolver os menus de navegação do jogo, utilizando os mesmos padrões e abordagens utilizadas desde então; (4) realizar testes com usuários e ajustes de balanceamento, se necessários; (5) lançar o primeiro capítulo do jogo em plataformas/*marketplaces* de jogos; e (6) criar e iniciar campanhas de divulgação com o objetivo de conseguir investimento para o projeto.

REFERÊNCIAS

BARROS, T.; SILVA, M.; ESPÍNOLA, E. State mvc: Estendendo o padrão mvc para uso no desenvolvimento de aplicações para dispositivos móveis. In: **Sexta Conferência Latino-Americana em Linguagens de Padrões para Programação**. [S.l.: s.n.], 2007.

COMMITTEE, I. S. C. *et al.* Ieee standard glossary of software engineering terminology (ieee std 610.12-1990). los alamitos. **CA: IEEE Computer Society**, v. 169, p. 132, 1990.

CROSS, N. **Design thinking: Understanding how designers think and work**. [S.l.]: Berg, 2011.

FAGAN, M. Design and code inspections to reduce errors in program development. In: **Software pioneers**. [S.l.]: Springer, 2002. p. 575–607.

GAMMA, E.; HELM, R.; JOHNSON, R.; VLISSIDES, J. Design patterns: Abstraction and reuse of object-oriented design. In: SPRINGER. **European Conference on Object-Oriented Programming**. [S.l.], 1993. p. 406–431.

KALINOWSKI, M.; TRAVASSOS, G. H. Uma abordagem probabilística para análise causal de defeitos de software. In: SBC. **Anais do XI Simpósio Brasileiro de Qualidade de Software**. [S.l.], 2012. p. 414–428.

KANODE, C. M.; HADDAD, H. M. Software engineering challenges in game development. In: IEEE. **2009 Sixth International Conference on Information Technology: New Generations**. [S.l.], 2009. p. 260–265.

MAIA, J. G. R. **CRAbGE: Uma arquitetura para motores gráficos flexíveis, expansíveis e portáveis para aplicações de realidade virtual,"**. [S.l.]: Departamento de Computação, Universidade Federal do Ceará, 2005.

MAIA, J. G. R.; VIDAL, C. A. Crabge: Um motor gráfico customizável, expansível e portável para aplicações de realidade virtual. In: **Proceedings of the VI Symposium on Virtual Reality**. [S.l.]: SBC/Faculdades COC, 2003. p. 3–14.

MELO, A.; ABELHEIRA, R. **Design Thinking & Thinking Design: Metodologia, ferramentas e uma reflexão sobre o tema**. [S.l.]: Novatec Editora, 2015.

NYSTROM, R. **Game programming patterns**. [S.l.]: Genever Benning, 2014.

RABIN, S. **Introduction To Game Development (Game Development)**. [S.l.]: Charles River Media, Inc., 2005.

SCHELL, J. **The Art of Game Design: A book of lenses**. [S.l.]: CRC press, 2008.

WIEBUSCH, D.; LATOSCHIK, M. E. Decoupling the entity-component-system pattern using semantic traits for reusable realtime interactive systems. In: IEEE. **2015 IEEE 8th Workshop on Software Engineering and Architectures for Realtime Interactive Systems (SEARIS)**. [S.l.], 2015. p. 25–32.

GLOSSÁRIO

B

Brainstorm: O brainstorming é uma dinâmica de grupo que é usada em várias empresas como uma técnica para resolver problemas específicos, para desenvolver novas ideias ou projetos, para juntar informação e para estimular o pensamento criativo..

F

Framework: Compilado de códigos de programação utilizado para um determinado fim dentro de alguma linguagem ou ferramenta de programação..

G

GDD: *Game Design Document*. Especificação de um jogo por via documental..

T

TBS: *Turn-based Strategy*.

V

VN: *Visual Novel*.