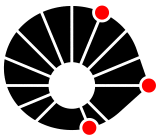

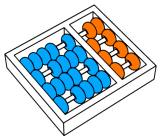

Double King: Design e Implementação de um Jogo de Xadrez com Regras Evolutivas e Mecânicas Customizáveis

Andreas Cisi Ramos *Emanuel Felipe Duarte*

Relatório Técnico - IC-PFG-25-25
Projeto Final de Graduação
2025 - Dezembro

UNIVERSIDADE ESTADUAL DE CAMPINAS
INSTITUTO DE COMPUTAÇÃO

Double King: Design e Implementação de um Jogo de Xadrez com Regras Evolutivas e Mecânicas Customizáveis

Andreas Cisi Ramos* Emanuel Felipe Duarte†

Resumo

Este trabalho apresenta o desenvolvimento do *Double King*, um jogo digital 2D que combina fundamentos do xadrez com princípios de game design contemporâneo, incorporando mecânicas evolutivas, geração procedural e regras customizáveis. O projeto investiga, de forma prática e teórica, como sistemas formais tradicionais podem ser estendidos por meio de transgressões sancionadas, produzindo novas formas de agência, incerteza e emergência. A implementação foi realizada em *Unity*, integrando um motor de xadrez compatível com variantes (*Fairy-Stockfish*), arquitetura modular, algoritmos de seleção procedural de peças e mecânicas inéditas como inventário, loja, peças evoluídas e o modo *Double King*. A análise do protótipo utiliza o arcabouço *Rules, Play, Culture* para discutir como essas escolhas sustentam *meaningful play* e reforçam a identidade *roguelike* do sistema. O resultado é um artefato que sintetiza competências técnicas e conceituais da graduação, demonstrando como a combinação entre teoria e prática pode gerar sistemas interativos ricos, expansíveis e alinhados a princípios fundamentais do design de jogos.

Palavras-chave: Game Design, Xadrez Evolutivo, Unity 2D, Sistemas Emergentes, Mecânicas Customizáveis.

*Instituto de Computação, UNICAMP, 13083-852 Campinas, SP. a246932@dac.unicamp.br
†Instituto de Computação, UNICAMP, 13083-852 Campinas, SP. emanuel@ic.unicamp.br

Sumário

1 Introdução

Desde as primeiras investigações formais sobre jogos, o Xadrez é utilizado como referência para compreender sistemas interativos estruturados por regras. A simplicidade das regras do Xadrez encobre um espaço combinatório amplo, tornando-o um domínio fértil para o estudo de tomada de decisão [1]. E com isso, se torna um exemplo de como regras simples, quando aplicadas de forma consistente, produzem profundidade estratégica, emergência e criam uma experiência de jogo significativa [2]. Essa tradição evidencia como jogos podem ser analisados enquanto sistemas formais nos quais estrutura, interação e consequência são elementos centrais.

Com base nesse entendimento, o jogo *Double King* foi desenvolvido como um sistema interativo que combina elementos do Xadrez com características do gênero *roguelike*, tais como geração procedural, alta rejogabilidade e imprevisibilidade estrutural. Essa abordagem foi uma subversão proposital da natureza formal do Xadrez. Sendo este um exemplo canônico de um Jogo de Estratégia Abstrata que, por definição, deve ter informação perfeita, onde todos os jogadores têm informação completa sobre a posição atual do tabuleiro [3]. O *Double King*, por sua vez, introduziu informação imperfeita e incerteza, permitindo ao jogador moldar parte da experiência, alterando as regras e influenciando o desenvolvimento do jogo de maneira ativa, criando um ambiente dinâmico, no qual as ações do jogador transformam continuamente a configuração do sistema.

Esse cenário oferece condições propícias para analisar como a combinação entre regras estáveis e evolutivas, baseado na modificação de mecânicas, sustenta diferentes formas de interação e estratégias, situando o *Double King* como um objeto relevante para discutir conceitos fundamentais de game design relacionados à emergência, à agência do jogador e à adaptação de sistemas formais.

2 Justificativa

A criação do *Double King* teve a oportunidade de articular teoria e prática em game design por meio do desenvolvimento de um artefato capaz de materializar os princípios investigados. Esse jogo permitiu examinar, de modo aplicado, como regras, sistemas emergentes e mecanismos de modificação estrutural se manifestam na experiência de jogo, ao mesmo tempo em que o processo de implementação consolida habilidades técnicas e competências adquiridas ao longo da graduação em Engenharia da Computação. Dessa forma, o projeto atuou simultaneamente como instrumento de investigação teórica do design de jogos e como síntese prática da formação acadêmica.

3 Fundamentação Teórica

Para orientar a análise conceitual e estruturar o estudo com base em referenciais consolidados, recorreu-se principalmente ao livro *Rules of Play: Game Design Fundamentals*, de Salen e Zimmerman (2003), que propõe um arcabouço teórico abrangente para o entendimento dos jogos enquanto sistemas formais, experiências de participação e artefatos

culturais.

3.1 Arcabouço Regras, Jogo e Cultura

O arcabouço teórico proposto por Salen e Zimmerman (2003) estrutura o entendimento dos jogos através de três esquemas primários de design [2, Cap.1: *What is this book about ?*]:

- **Rules (Regras):** foca na organização do sistema projetado, nas estruturas lógicas e matemáticas e nas qualidades formais do jogo. Essas são as estruturas que regem a identidade do sistema.
- **Play (Jogabilidade/Experiência):** foca na experiência humana do sistema, na participação do jogador, no prazer, no significado e na interação.
- **Culture (Cultura):** foca nos contextos mais amplos em que o jogo está inserido, como as relações entre o jogo e os contextos culturais, ideologias e valores.

3.2 Fundamentos de Game Design

A partir do arcabouço teórico proposto por Salen e Zimmerman (2003), destacam-se os seguintes conceitos fundamentais para o design de jogos:

- **Meaningful Play (Experiência de jogo significativa):** conceito central que orienta os designers na criação de experiências que tenham sentido e relevância para os jogadores, sendo o objetivo fundamental do design de jogos bem-sucedido [2, Cap.3: *Meaningful Play*].
- **Lusory Attitude (Atitude lusória):** estado mental em que o jogador aceita voluntariamente regras, objetivos e obstáculos arbitrários, possibilitando a existência e o funcionamento do jogo [2, Cap.7: *Defining Games*].
- **Emergent Systems (Sistemas emergentes):** sistemas que produzem complexidade e resultados imprevisíveis a partir de um conjunto de regras simples [2, Cap.14: *Games as Emergent Systems*].

3.3 Conceitos Complementares

- **Playtesting (Teste de Jogabilidade):** componente fundamental do design iterativo, envolvendo um ciclo contínuo de criação, teste e revisão [2, Cap.2: *The Design Process*].
- **Roguelike (Gênero Roguelike):** subgênero inspirado em *Rogue* (1980), cuja definição, embora ainda debatida, geralmente inclui características como geração procedural, elevada rejogabilidade e o conceito de morte definitiva. Essas propriedades reforçam a essência do gênero, baseada na variabilidade extrema e no aprendizado emergente ao longo de múltiplas tentativas [4].

3.4 Engine Unity e Conceitos Fundamentais do Desenvolvimento 2D

A *Unity* [5] é uma *engine* de desenvolvimento amplamente utilizada para a criação de jogos digitais, oferecendo ferramentas visuais, um ecossistema modular baseada em C#. Seu modelo operacional adota uma arquitetura centrada em *GameObjects* e *Components*, na qual comportamentos podem ser acoplados, removidos e combinados de forma dinâmica.

- **GameObject (Objeto do Jogo):** Unidade estrutural fundamental da *Unity* e representa qualquer entidade presente em uma cena. Um *GameObject* funciona como um contêiner que recebe funcionalidade através da adição de *Componentes*.

- **Components (Componentes):** Elementos que definem comportamento, aparência e funcionalidades de um *GameObject*. A lógica personalizada do jogo também é implementada como componentes, por meio de scripts em C#.

- **Scripts C# (Scripts em C#):** Componentes que implementam regras, decisões, interações e fluxos internos do jogo.

- **Sprites:** Imagens 2D utilizadas como recursos gráficos para exibição na tela. Eles representam texturas individuais importadas como objeto e são renderizados através do componente *SpriteRenderer*, que controla como essas imagens aparecem na cena.

- **ScriptableObjects (Objetos Scriptáveis):** São estruturas de dados reutilizáveis independentes do ciclo de vida dos *GameObjects*. Eles permitem armazenar configurações, valores, tabelas, curvas, parâmetros de balanceamento e perfis sem necessidade de instanciação na cena.

- **UnityEvent e Arquitetura Event-Driven (Orientada a Eventos):** Sistema nativo da *Unity* para implementar comunicação desacoplada entre objetos. Ele permite que *GameObjects* e componentes se comuniquem sem depender diretamente uns dos outros, seguindo o paradigma direcionado a eventos.

4 Objetivos

O objetivo central desse trabalho teve como princípio investigar e documentar o processo criativo no design de jogos, com foco na aplicação de técnicas e conceitos fundamentais, utilizando o embasamento teórico e a experimentação prática. Nesse contexto, o jogo *Double King* foi desenvolvido como um artefato de design capaz de materializar princípios fundamentais do game design, permitindo examinar como variações de regras, pela criação de mecânicas, e decisões de implementação tornaram a experiência de jogo significativa. O projeto igualmente se propôs a sintetizar os conhecimentos adquiridos na graduação, integrando habilidades técnicas e fundamentos teóricos desenvolvidos ao longo do curso, aplicando-os na construção de um jogo como um exemplo de software.

4.1 Objetivos específicos

4.1.1 Construção de Base Técnica Sólida

Antes de explorar mecânicas evolutivas, tornou-se fundamental estabelecer uma base estrutural robusta tomando o xadrez como sistema formal de referência. Essa base deve ser modular e extensível, garantindo estabilidade para futuras adições de regras dinâmicas, incluindo a definição da arquitetura essencial do jogo (como tabuleiro, peças e lógica central), a implementação dos elementos necessários para a interação em ambiente digital e a integração de um oponente artificial capaz de sustentar o conflito próprio do sistema. Essa etapa estabeleceu o alicerce sobre o qual as demais camadas de design poderão ser investigadas.

4.1.2 Design de Novas Mecânicas e Exploração Teórica

O desenvolvimento do sistema buscou explorar mecânicas que ampliaram as regras clássicas do xadrez e permitiram avaliar, sob a ótica teórica, como o arcabouço *Rules, Play e Culture*, como adição de mecânicas favoreceram a experiência do jogo significativa. A partir de uma base estrutural estável, investigou-se um conjunto de possibilidades teóricas oferecidas pela literatura, examinando como variações de regras, modificações temporárias e estruturas emergentes foram incorporadas ao jogo para expandir sua expressividade. O objetivo não foi determinar previamente quais mecânicas são mais eficazes, mas explorar, analisar e prototipar soluções que demonstraram potencial para produzir interações significativas.

4.1.3 Documentação Crítica e Síntese Formativa

O processo de desenvolvimento, as decisões de design, as justificativas conceituais e as transformações na arquitetura constituíram parte essencial do projeto. Essa documentação articulou o percurso teórico-prático, relacionando escolhas de implementação aos princípios de game design estudados, ao mesmo tempo em que o artefato consolidou o projeto como uma síntese das competências desenvolvidas na graduação. O resultado final configura-se como um protótipo expansível, apto a servir tanto como artefato de investigação quanto como base para evolução posterior em projetos futuros.

5 Metodologia

A metodologia adotada para o desenvolvimento do *Double King* combinou investigação teórica, fundamentação técnica, criação artística e ciclos iterativos de experimentação. O processo, de natureza não linear, estruturou-se em eixos que dialogam entre si ao longo do processo — revisão conceitual, modelagem, ideação, implementação, avaliação e refinamento — permitindo que decisões teóricas, técnicas e estéticas evoluíssem simultaneamente ao longo do projeto.

5.1 Revisão Bibliográfica e Fundamentação Técnica

A primeira etapa consistiu na revisão de literatura para reunir modelos conceituais, exemplos práticos e princípios de design que orientaram a criação do jogo. Esse estudo envolveu obras fundamentais como *Rules of Play* e *Game Design Workshop* [6], além da análise de literatura cinza e de títulos contemporâneos, como *The Balatro Timeline* [7], que demonstram como pequenas alterações em sistemas tradicionais podem expandir a profundidade estratégica e gerar experiências altamente significativas. O objetivo dessa revisão não foi definir antecipadamente quais mecânicas seriam adotadas, mas construir um repertório capaz de orientar decisões ao longo do processo e fornecer referências de como jogos baseados em regras podem produzir *meaningful play*.

Em paralelo, desenvolveu-se uma fundamentação técnica voltada ao domínio da *Unity*, visando envolver a aprendizagem progressiva dos recursos centrais da *engine* para viabilizar a implementação das ideias de design. A *Unity* mostrou-se particularmente adequada por sua ampla adoção na indústria, pela documentação extensa e pela arquitetura baseada em *GameObjects* e *Components*, que favorece modularidade e extensão. Esse estudo técnico proporcionou a infraestrutura necessária para desenvolver sistemas dinâmicos e iterar sobre eles com agilidade ao longo do projeto.

5.2 Construção da Base Técnica

A construção técnica iniciou-se pela definição de uma base estrutural sólida sobre a qual as demais mecânicas poderiam ser desenvolvidas. O xadrez foi adotado como sistema inicial, servindo como estrutura estável e bem definida para experimentações futuras. A arquitetura do jogo foi projetada com foco em modularidade e baixo acoplamento, garantindo que peças, movimentos e regras pudessem ser estendidos ou substituídos sem interferência entre os componentes.

Para que essa base fosse funcional como jogo, era necessário preservar o conflito central previsto no xadrez, o que exigia a presença de um oponente consistente e autônomo. O motor *Fairy-Stockfish* [8] foi um elemento central na construção desse oponente artificial, permitindo processar a configuração do tabuleiro e gerar decisões estratégicas mesmo quando o sistema inclui peças com novos padrões de movimento ou tamanhos de tabuleiros não convencionais.

Essa escolha tornou-se especialmente relevante tendo em vista que um oponente capaz de antecipar e compreender todas as futuras regras evolutivas ou mecânicas customizáveis extrapolaria o escopo do projeto, exigindo modelos de decisão complexos. Diante disso, a pesquisa voltou-se para a busca de um motor que operasse não apenas sobre as regras tradicionais do xadrez, mas que ainda oferecesse flexibilidade suficiente para lidar com extensões estruturais.

5.3 Ideação e Prototipação das Novas Mecânicas

A ideação seguiu princípios descritos em *Game Design Workshop: A Playcentric Approach to Creating Innovative Games* [6], segundo os quais a clareza da mecânica e da experiência

desejada deve preceder a produção digital. Para isso, mecânicas foram inicialmente exploradas por meio de esboços e diagramas, permitindo avaliar potenciais desdobramentos sem o custo e a rigidez que a implementação pode impor.

Essa etapa buscou identificar como alterações de regras, diferentes modos de jogo ou modificações no comportamento das peças poderiam expandir o espaço de agência do jogador. Assim, reduzir o tempo de iteração de cada validação conceitual, favorece a eliminação de ideias desalinhadas aos objetivos do projeto [6].

5.4 Avaliação Prática e Refinamento Iterativo

O *playtesting with confidants*, conceito definido por Fullerton (2024) como teste de jogabilidade com confidentes, foi o tipo de avaliação prática explorada neste trabalho com a ajuda de colegas e familiares. Esse tipo de testador é recomendado para as fases iniciais do desenvolvimento, quando o objetivo principal é verificar legibilidade, clareza e compreensão das regras sem a necessidade de um protótipo completo [6].

Durante as sessões, adotou-se a prática de solicitar que os participantes "pensassem em voz alta", permitindo registrar suas expectativas, dúvidas, hesitações e interpretações espontâneas. Esse procedimento, recomendado por Fullerton (2024), fornece acesso direto ao raciocínio do jogador, revelando antecipações incorretas, confusões na interface e discrepâncias entre intenção do design e percepção do usuário.

O feedback coletado foi registrado tanto em forma qualitativa, incluindo impressões verbais, dúvidas, interpretações e comentários sobre usabilidade, quanto em dados quantitativos simples, como duração das partidas e frequência de erros de operação. Essas informações foram anotadas e utilizadas como base para identificar inconsistências de design, revisar regras, ajustar comportamentos do sistema e orientar novas ideias durante o processo iterativo.

O refinamento do projeto emergiu diretamente desses ciclos contínuos de experimentação, teste e revisão. Em vários casos, ideias foram descartadas ainda na fase conceitual ou imediatamente após as primeiras sessões de teste, evitando desperdício de tempo e concentrando esforços nas mecânicas com maior potencial. Esse processo iterativo permitiu ajustar regras, reequilibrar sistemas e direcionar a evolução do protótipo para novas mecânicas com base na resposta real dos jogadores.

5.5 Criação da Identidade Visual

O desenvolvimento da identidade visual ocorreu paralelamente às etapas técnicas, uma vez que a arte exerce papel central na legibilidade e na comunicação das ações do jogador. Obras como *The Art of Cuphead* (2020) destacam que a clareza visual e a coerência estética são essenciais para evitar ambiguidade mecânica e reforçar a intuição durante a jogabilidade.

Com base nesses princípios, optou-se pela criação de pixel art autoral utilizando o LibreSprite [10], um *software open source* voltado para arte 2D. A ferramenta permitiu produzir elementos visuais simples e expressivos, capazes de comunicar função e estado de maneira imediata, além de estabelecer uma identidade estética consistente para o projeto.

5.6 Documentação e Conclusões

Essa etapa se preocupou no registro de decisões de design, alterações na arquitetura, justificativas conceituais e sucessivas revisões do sistema. Esse registro garantiu transparência ao processo metodológico e forneceu a base necessária para a análise crítica realizada ao final do projeto.

As conclusões emergem da comparação entre os objetivos inicialmente definidos, as escolhas efetuadas ao longo do processo e os impactos observados na experiência do jogador. Incluem também reflexões sobre mecânicas que surgiram durante o percurso, algumas implementadas, outras apenas conceituadas e esboçadas, cuja viabilidade só se revelou graças à interação contínua entre fundamentação teórica, prática técnica e ciclos iterativos de experimentação e avaliação.

6 Desenvolvimento da Arquitetura Base

6.1 Base Sólida do Xadrez

Para estabelecer uma base sólida e extensível do Xadrez tradicional, foi desenvolvida uma arquitetura seguindo o padrão MVC (Modelo-Visão-Controlador) adaptado para *Unity*. O sistema foi projetado com foco em separação de responsabilidades, onde cada componente pode ser modificado ou estendido sem afetar o funcionamento do sistema como um todo.

6.1.1 Arquitetura de Controladores

A arquitetura utilizou um conjunto de controladores que formam a base estrutural do jogo. O *GameManager* atua como orquestrador central, mantendo referências para todos os controladores e coordenando o estado geral do jogo através de um sistema de eventos (*event-driven*). Os principais controladores criados que compõem essa base são:

- **GameManager**: Gerencia estados do jogo, coordena inicialização e transições entre fases. Implementando o padrão *Singleton* para acesso global e centraliza a comunicação entre todos os controladores.
- **BoardController**: Responsável pela criação, destruição e gerenciamento do tabuleiro, incluindo validação de movimentos legais e detecção de estados de xadrez (xeque, xeque mate, empate), mantendo referência ao modelo *Board* e gerenciando a matriz de *Tile*s.
- **PieceController**: Gerencia criação, destruição e posicionamento de peças, mantendo listas separadas para peças do jogador e do oponente, coordenando a instanciação de *GameObject*s de peças.
- **PlayerController**: Controla ações do jogador e validação de turnos, gerenciando e coordenando as ações do jogador.
- **ChessController**: Coordena a lógica de turnos e repassa comandos ao sistema do oponente artificial quando necessário.

6.1.2 Arquitetura de Modelos

A camada de Modelo contém as estruturas de dados fundamentais que representam o domínio do jogo, contendo os seguintes *scripts* desenvolvidos:

- **Board (Tabuleiro)**: Representa o tabuleiro como uma matriz bidimensional de *Tile*s, armazenando largura e comprimento.
- **Tile (Casa)**: Representa uma casa do tabuleiro, contendo referência à peça posicionada, coordenadas (x, y), posição no mundo *Unity* e estado visual. Cada *Tile* expõe um tipo (*TileAction*) que define sua apresentação (*sprite*) e é ajustado via `ChangeTileAction`: *Idle*, *Movement*, *Selected*, *Capture*, *SpecialMove*. Quando clicada os controladores reagem decidindo o comportamento com base nesse *TileAction*.
- **Piece (Peça)**: Representa uma peça de xadrez, contendo tipo(Rei, Rainha, Torre, Bispo, Cavalo, Peão), cor, posição, lista de movimentos acoplados, implementando métodos para cálculo de movimentos disponíveis.
- **PieceDescription (Descrição de Peça)**: Estrutura de dados que descreve uma peça de forma serializável, contendo tipo, cor e movimentos, permitindo criação de layouts de oponente e configurações de peças.
- **IPieceMovement e Implementações**: Interface e classes concretas (*KingMovement*, *QueenMovement*, *RookMovement*, *BishopMovement*, *KnightMovement*, *PawnMovement*) que implementam padrões de movimento específicos.

6.1.3 Geração Dinâmica de Tabuleiro

O sistema de tabuleiro foi implementado permitindo possíveis variações de tamanho. A geração é realizada pelo *BoardController* através do método `CreateBoard(int boardWidth, int boardLength)`, que cria uma matriz de *Tile*s com as dimensões especificadas.

O processo de criação envolveu o cálculo dinâmico de escala dos *tiles*, calculado proporcionalmente ao tamanho do tabuleiro com um limite mínimo para manter legibilidade em tabuleiros grandes. O método `CreateTileGrid` cria uma grade de *GameObjects* com *tiles* diferentes para bordas (esquerda e direita) e centro, posicionando cada *tile* com base em coordenadas calculadas. Cada *tile* recebe um *sorting order*, valor numérico usado pela *Unity* para determinar a ordem de renderização de *sprites* em 2D, onde objetos com valores maiores são renderizados por cima de objetos com valores menores. Assim, as *tiles* da frente recebem um *sorting order* maior, criando a sensação de profundidade mesmo em um jogo 2D.

6.1.4 Sistema de Peças e Movimentos Acopláveis

As peças foram implementadas como entidades que podem receber múltiplas funções de movimento e ataque acopladas dinamicamente. O sistema utiliza o padrão *Strategy* através da interface *IPieceMovement* e da classe abstrata *PieceMovementBase*. O padrão *Strategy*

é um padrão de design comportamental que permite definir uma família de algoritmos, encapsulá-los e torná-los intercambiáveis. Neste contexto, cada tipo de movimento (Rei, Rainha, Torre, etc.) é uma estratégia que pode ser acoplada dinamicamente a uma peça, permitindo que o comportamento de movimento seja selecionado em tempo de execução sem modificar a estrutura da classe *Piece*.

Interface IPieceMovement: Define o contrato que todos os movimentos devem implementar:

- `GetAvailableMoves`: Retorna os movimentos disponíveis da peça, podendo combinar movimentos e ataque;
- `GetMovesPattern`: Retorna apenas padrões de movimento;
- `GetAttackPattern`: Retorna apenas padrões de ataque.

Com essa interface foi possível obter padrões de movimento e ataque diferentes, como o do Peão que por mais que se move apenas uma casa para frente, pode capturar peças inimigas diagonalmente. Cada tipo de peça possui uma classe que herda de *PieceMovementBase* e implementa seus padrões específicos. A classe *Piece* mantém uma lista `List<IPieceMovement> Movements` que permite adicionar múltiplos movimentos à mesma peça. Isso permitiu que uma peça tenha, por exemplo, movimentos de Torre e Cavalo simultaneamente, criando comportamentos híbridos que podem ser adicionados em tempo de execução através do método `AddPieceMovement(PieceType extraMovementType)` [Figura 1].

```
public void AddPieceMovement(PieceType type)
{
    switch (type)
    {
        case PieceType.King:    Movements.Add(new KingMovement());    break;
        case PieceType.Queen:   Movements.Add(new QueenMovement());   break;
        case PieceType.Rook:    Movements.Add(new RookMovement());    break;
        case PieceType.Bishop:  Movements.Add(new BishopMovement());  break;
        case PieceType.Knight:  Movements.Add(new KnightMovement());  break;
        case PieceType.Pawn:    Movements.Add(new PawnMovement());    break;
        default: break;
    }
}
```

Figura 1: Métodos para acoplar novos padrões de movimento a uma peça (*Piece.cs*).

6.2 Oponente Artificial

Uma arquitetura para o sistema de oponente artificial foi desenvolvida utilizando o padrão de interface para abstrair a integração da *engine* de xadrez sem modificar o código do jogo.

6.2.1 Interface IChessEngine

A interface *IChessEngine* define o contrato que a *engine* deve seguir, abstraindo detalhes específicos de cada engine. Os principais métodos incluem:

- `InitializeAsync()`: Inicializa o *engine* de forma assíncrona;
- `SetPositionAsync(BoardPosition position)`: Define a posição atual do tabuleiro;
- `CalculateMoveAsync(CalculationConstraints constraints)`: Solicita o cálculo da melhor jogada.

6.2.2 Implementação com FairyStockfish

A solução implementada utilizou o *FairyStockfish*, um *engine* que suporta variantes do xadrez e tabuleiros de tamanhos não-padrão. A classe *FairyStockfishEngine* implementa *IChessEngine* e utiliza um componente *UCIEngine* para comunicação via protocolo UCI (*Universal Chess Interface*), que permite comunicação padronizada com *engines* de xadrez através de comandos de texto.

O *FairyStockfishEngine* gerencia a inicialização do processo do *engine*, a configuração de variantes através de arquivos `.fv` (fairy variant) gerados dinamicamente, o ajuste de parâmetros do *engine* (*threads*, profundidade) e a conversão entre formatos Unity e UCI.

O arquivo de variante fairy permite configurar algumas regras tradicionais do jogo, incluindo: definição do tamanho do tabuleiro; definição de peças canônicas através de `pieceTypes` usando notação específica do *Fairy-Stockfish*; criação de peças com novos padrões de movimento; e configuração de regras especiais como promoção de peões, *en passant*, roque e duplo passo do peão.

6.2.3 Conversão de Estados

A classe *BoardStateConverter* realiza a tradução bidirecional entre o estado do tabuleiro em *Unity* e a notação FEN (*Forsyth-Edwards Notation*), que é o formato padrão usado por *engines* de xadrez, implementa a conversão de coordenadas UCI para posições do tabuleiro Unity através do método `UCIMoveToUnity`, que recebe uma string UCI (por exemplo, `"bestmove e2e4"`, indicando que a engine recomenda mover a peça de `e2` para `e4`) e retorna uma tupla contendo as posições de origem e destino em `Vector2Int`, além de informações sobre promoção de peão, quando aplicável [Figura 2].

```
var (from, to, promotion) = BoardStateConverter.UCIMoveToUnity(bestMoveUCI);

Piece piece = boardController.GetPieceAtPosition(from);

if (piece != null)
{
    var validMoves = piece.GetAvailableMoves(boardController.CurrentBoardState);

    if (validMoves.Contains(to))
    {
        boardController.MovePiece(piece, to);
    }
    else
    {
        Debug.LogWarning("Movimento sugerido pelo engine é inválido no estado atual.");
    }
}
```

Figura 2: Conversão *UCI* para coordenadas *Unity* e aplicação do movimento (método *UCIMoveToUnity*).

Esse código demonstra a integração prática entre o comando retornado pela *engine* e o sistema de movimentação de peças. Assim, o motor de xadrez calcula a jogada como texto, que é então convertido para posições internas e validado/invocado pelas regras e classes de movimento do *Unity*.

6.2.4 Controlador do Oponente Artificial

O *BotController* gerencia o fluxo de cálculo assíncrono do oponente artificial, validando o estado do tabuleiro antes de solicitar cálculo, gerenciando atrasos configuráveis para simular tempo de pensamento, validando movimentos retornados pelo *engine* contra o estado atual do tabuleiro, implementando sistema de retry e re-sincronização em caso de erros e reinicializando o *engine* automaticamente se necessário.

A interface de comunicação foi projetada de forma simples e direta: o *BotController* apenas solicita a melhor jogada para um estado específico do tabuleiro através de `CalculateMoveAsync`, abstraindo toda a complexidade da comunicação com o *engine*. O resultado é retornado via evento *OnMoveCalculated* e aplicado ao tabuleiro.

6.2.5 Limitações da Engine de Xadrez

O oponente artificial utilizado no projeto calcula suas jogadas exclusivamente a partir do estado atual do tabuleiro, operando apenas como um oponente que conhece uma versão expandida das regras do xadrez tradicional. Como esse motor não consegue interpretar ou avaliar regras alternativas, o que exigiria algum treinamento após a criação de mecânicas extras, essa limitação passou a orientar diretamente o design do jogo.

A assimetria entre um inimigo estável, limitado ao uso de regras tradicionais, e um jogador capaz de explorar futuras outras regras e mecânicas passou a orientar a ideia central do jogo. Conceito que será aprofundado no tópico seguinte, ao apresentar as Regras Evolutivas e as Mecânicas Customizáveis.

7 Regras Evolutivas e Mecânicas Customizáveis

7.1 Mecânicas Principais

7.1.1 Níveis de Dificuldade

Visando explorar de maneira significativa a diferença entre o oponente artificial e o jogador, adotou-se a assimetria no número de peças como elemento central do design. A desvantagem numérica enfrentada pelo jogador foi estruturada como parte intencional do desafio, reforçando que todo jogo opera sobre um conflito artificial [2, Cap.20: *Games as Systems of Conflict*]. Nesse contexto, o jogador aceita voluntariamente tais restrições por meio da Atitude Lusória, reconhecendo obstáculos arbitrários como condições legítimas do sistema.

A partir dessa lógica, utilizou-se uma mecânica característica de jogos *roguelike*, estruturando o sistema em níveis sucessivos de dificuldade crescente. Jogos como *Balatro* reforçam esse formato ao combinar aleatoriedade, adaptação estratégica e ciclos de dificuldade incremental [7]. Dessa forma, cada vitória (xeque mate) conduz o jogador a tabuleiros maiores e a conjuntos inimigos mais numerosos, refletindo o modelo de progressão escalonada e desafios acumulativos típico do gênero.

Além da estrutura mecânica, buscou-se adicionar uma camada narrativa para reforçar a escalada de dificuldade. Cada estágio apresenta um *Boss* (chefe) baseado em uma peça de xadrez, progredindo de acordo com seus valores: do peão ao rei. Isso transforma a progressão em uma sequência temática clara, na qual o jogador enfrenta partes do exército adversário de forma simbólica.

Essa progressão vai além do que Salen e Zimmerman (2003) definem como Sistema de Conflito, pois a assimetria crescente entre os exércitos pode tornar o desafio matematicamente impossível, comprometendo a sensação de justiça, elemento essencial para um conflito equilibrado que disperte *meaningful play* [2, Cap.20: *Games as Systems of Conflict*]. Em contrapartida, essa limitação abre espaço para novas mecânicas que devolvem a capacidade de agir do jogador, oferecendo novos caminhos em um cenário adverso.

Detalhes da Implementação A mecânica de dificuldade crescente foi implementada através do *LevelController*, que gerencia a progressão de níveis com tabuleiros progressivamente maiores e exércitos crescentes de peças. O controlador mantém um dicionário de configurações de nível (*LevelConfig*), que contém os seguintes parâmetros:

- *Difficulty*(Dificuldade): Enumeração que identifica o tipo de exército (*Pawn, Knight, Bishop, Rook, Queen, King*);
- *boardWidth* e *boardLength*: Dimensões do tabuleiro para o nível;

- *opponentLayout*: Matriz *PieceDescription[,]* que define o exército do oponente;
- *description*: Descrição textual do nível.

O sistema foi projetado para facilitar ajustes de balanceamento. Novos níveis podem ser adicionados simplesmente criando novas entradas no dicionário *levelConfigs*, exemplo na Figura 3, e os layouts de oponente são gerados através de métodos dedicados (*CreatePawnDifficultyLayout*, *CreateKnightDifficultyLayout*, etc.), como mostra a Figura 4, permitindo ajustes rápidos na composição dos exércitos.

```
levelConfigs[1] = new LevelConfig
{
    difficulty = Difficulty.Pawn,
    boardWidth = 5,
    boardLength = 5,
    description = "Basic chess setup - Pawn difficulty",
    opponentLayout = CreatePawnDifficultyLayout()
};
```

Figura 3: Exemplo de configuração de *LevelConfig* para o nível de dificuldade do Peão.

```
private PieceDescription[,] CreatePawnDifficultyLayout()
{
    PieceDescription[,] layout = new PieceDescription[2, 5];

    layout[0, 2] = new PieceDescription(PieceType.King, PieceColor.Black);

    for (int i = 0; i < 5; i++)
    {
        layout[1, i] = new PieceDescription(PieceType.Pawn, PieceColor.Black);
    }

    return layout;
}
```

Figura 4: Exemplo de *CreatePawnDifficultyLayout*: geração do *layout* do exército do oponente para o nível de dificuldade do Peão.

Exemplo dos nível de dificuldade do Peão implementados dentro da *Unity* podem ser vistos na Figura 5.

7.1.2 Double King Mode

Considerando que o jogo adota dificuldade crescente como eixo central da experiência, tornou-se necessário resolver como lidar com situações de empate sem comprometer a progressão. A primeira solução testada foi tratar o empate como vitória automática do jogador,

Figura 5: Exemplo ilustrativo do nível de dificuldade Peão.

permitindo avançar para o próximo nível de dificuldade. No entanto, durante as primeiras sessões de *playtesting*, observou-se que essa abordagem alterava a dinâmica estratégica desejada: os jogadores passaram a buscar o empate de forma deliberada, usando trocas de peças como caminho mais fácil para encerrar a partida. Isso reduzia a importância do xeque-mate e enfraquecia a base estratégica do xadrez tradicional.

Identificado que o empate não poderia funcionar como parte da estratégia de evolução, tornou-se necessário desenvolver um mecanismo específico para resolver essa condição sem prejudicar o ritmo do jogo. Essa avaliação também abriu espaço para integrar uma mecânica customizável e distinta do xadrez tradicional. A solução deveria preservar o desafio, impedir a exploração do sistema e introduzir um momento decisivo que mantivesse a coerência com a experiência proposta.

Dessa necessidade surgiu um modo exclusivo: o modo *Double King*, um sistema de desempate que combina decisão estratégica e sorte. Nesse modo, um empate desencadeia uma disputa direta entre os dois reis no mesmo tabuleiro. A resolução ocorre por meio de ciclos alternados de remoção de *tiles* e movimentação: (1) O rei inimigo seleciona uma *tile* que será removida; (2) O jogador então realiza seu movimento sem saber qual *tile* foi marcada; se escolher exatamente essa *tile*, o modo de desempate termina e o jogador perde; (3) Caso contrário, a rodada é passada para o jogador, que escolhe uma *tile* para ser eliminada; (4) O rei inimigo faz seu movimento, novamente sem saber qual *tile* foi escolhida — e o ciclo se repete até que um dos reis caia do tabuleiro e seja eliminado.

A introdução de elementos de incerteza no modo *Double King* foi fundamental para tornar o desempate significativo. O *meaningful play* depende de sistemas que não sejam totalmente previsíveis [2, Cap.15: *Games as Systems of Uncertainty*]. O modo combina agência do jogador, ao permitir que o jogador escolha qual *tile* será removida, com risco

e incerteza, já que o movimento do rei adversário ocorre sem conhecimento prévio da *tile* marcada. Assim, o jogador pode influenciar a situação, mas nunca determiná-la por completo, mantendo um equilíbrio entre controle parcial e incerteza que torna cada rodada tensa, imprevisível e relevante.

Nos *playtests*, essa dinâmica mostrou-se eficaz ao motivar rejogabilidade. Os participantes relataram que desejariam experimentar esse modo novamente, mesmo sem relação com a progressão principal, buscando manipular o risco e antecipar o comportamento do oponente. Isso indica que o *minigame* não apenas evita empates, mas também sustenta engajamento próprio ao explorar a relação equilibrada entre agência, probabilidade e incerteza.

Uma visualização do modo *Double King* em execução pode ser vista em [Figura 6].

Figura 6: Modo *Double King* em execução na rodada de movimentação do jogador para o desempate do nível de dificuldade Bispo.

Detalhes da Implementação O modo é ativado quando um empate é detectado. O *DoubleKingModeController* inicializa o modo removendo todas as peças do tabuleiro e reposicionando os reis para o centro da primeira e última fileira. Os outros métodos dessa classe ficam responsáveis pela remoção de *tiles* ou pela movimentação do rei, alternando a rodada entre jogador e oponente ao fim da ação. Destaca-se aqui a ideia de configurar o tabuleiro como uma matriz de *tiles* em que cada *tile* é um *GameObject* independente. Dessa forma, cada *tile* pode ser removida individualmente sem quebrar ou refatorar o tabuleiro inteiro [Figura 7].

```
public void OnKingMoved(Tile newTile)
{
    if (tileToRemove != null && newTile != null &&
        newTile.position == tileToRemove.position)
    {
        Piece fallenKing = newTile.piece;
        StartCoroutine(KingAndTileFallSequence(fallenKing, newTile));
        tileToRemove = null;
        return;
    }

    boardController.RemoveTile(tileToRemove.position);
    ChangeRemoveTurn();
    tileToRemove = null;

    if(isPlayerRemoveTurn){
        StartPlayerTileRemovalTurn();
    }
    else{
        StartOpponentTileRemovalTurn();
    }
}
```

Figura 7: Método executado após movimentação do rei, responsável pela remoção de *tile* e alternância de turno; se o rei estiver na *tile* marcada, executa-se a queda do rei (método *OnKingMoved*).

Inicialmente, foi testado o *FairyStockfish* para calcular jogadas no *Double King Mode*. No entanto, os primeiros testes mostraram inconsistências no comportamento do oponente artificial: em alguns casos o rei não se movia, em outros tentava deslocar-se para uma *tile* já removida, ou simplesmente repetia movimentos sem qualquer lógica estratégica. Isso ocorre porque os motores de variantes assumem um tabuleiro retangular contínuo, sem buracos, e tratam configurações com apenas dois reis como empate. Assim, o *engine* era incapaz de interpretar corretamente as regras do modo e não conseguia operar dentro da incerteza gerada pela remoção dinâmica de *tiles*.

Vale ressaltar que no *FairyStockfish* é possível criar uma variante do xadrez onde são definidas peças com padrões customizados de movimento, sendo teoricamente possível incluir uma peça totalmente estática — sem ataque e sem possibilidade de deslocamento — para representar buracos como obstáculos fixos no tabuleiro. No entanto, os resultados permaneceram inconsistentes: o *engine*, ao não identificar movimentos válidos que pudessem levar a uma condição de vitória com apenas dois reis, deixava de responder, ocasionando *timeout* mesmo após múltiplas tentativas de *retry*.

Diante dessas limitações, tornou-se necessário desenvolver um algoritmo próprio para esse modo, tanto para a movimentação, escolha e remoção de *tiles*. Para suprir essa demanda, foi criado um algoritmo simples baseado em seleção probabilística ponderada. O

método *ChooseWeightedByDistance* (Apêndice A) calcula pesos para cada *tile* candidata com base na distância ao rei adversário:

- Durante a fase de remoção de *tiles* pelo oponente: prioriza *tiles* próximas ao rei do jogador, gerando escolhas mais agressivas.
- Durante a fase de movimentação do rei inimigo: a lógica inversa busca afastá-lo do rei do jogador durante a fase de exclusão de *tiles*.
- Os pesos são normalizados e uma amostragem proporcional ocorre via *ChooseIndexByWeights* (Apêndice A), garantindo que *tiles* com maior peso tenham maior probabilidade de serem selecionadas, mas preservando um elemento de sorte.

O resultado é um algoritmo que não busca retornar a melhor jogada possível, mas busca equilibrar a estratégia e a probabilidade. Em certas situações, pode ser vantajoso isolar o rei inimigo; em outras, mover-se de forma aparentemente aleatória pode gerar posições mais favoráveis. Assim, adotou-se uma solução híbrida: incorpora algum grau de estratégia, mas permanece ancorada em probabilidade e variabilidade. Outras soluções mais sofisticadas poderiam ser exploradas, mas esta abordagem demonstrou ser suficientemente robusta e proporcionou um desafio relevante para o desenvolvimento.

7.2 Quebrando as Regras Passivamente

Diante de um oponente que opera com as regras tradicionais e uma grande diferença de valor entre as peças, o conflito no *Double King* rapidamente se torna estruturalmente injusto. Isso levou, de maneira natural, à necessidade de introduzir mecânicas capazes de quebrar a rigidez formal do sistema, algo que Salen e Zimmerman (2003) descrevem como Transgressão Sancionada, incorporando mecânicas que se comportam como trapaças na visão do oponente artificial. Se elementos fundamentais do jogo, como o número de peças, seus movimentos ou até mesmo a ordem de turno, pudessem ser explorados para enfrentar a assimetria crescente, então o design precisava oferecer ao jogador os meios necessários para a essa quebra de regra subvertendo as regras tradicionais do xadrez.

A partir dessa premissa, um sistema no qual o jogador tivesse controle sobre as peças que utiliza pareceu uma solução natural. Essa ideia de peças como elementos configuráveis aproxima o *Double King* da estrutura de jogos de construção estratégica, como *Balatro* [7], um jogo *roguelike* focado no Poker, no qual o poder do jogador não é dado de imediato, mas construído progressivamente. Dessa forma, o jogo se iniciando apenas com o rei, permitindo que o jogador adquira e selecione novas peças ao longo de sua jornada se alinha com esse princípio.

Com base nessa lógica, duas mecânicas se tornam consequências diretas do design: (1) um Inventário, no qual o jogador organiza as peças conquistadas, decidindo quais utilizar e quais preservar, o que depende de uma mecânica de posicionamento das peças no início de cada desafio; e (2) um sistema de Loja ou Obtenção de Peças, responsável pela aquisição de novas peças e possíveis habilidades adicionais.

Essas mecânicas ampliam a capacidade do jogador de subverter o sistema, mas fazem isso antes ou depois da partida começar, preparando o terreno estratégico sem alterar diretamente as regras durante a partida de xadrez. Assim, no contexto deste projeto, foram estabelecidas duas categorias de mecânicas: (1) "quebras de regra passivas", criadas para organizar as camadas de intervenção no sistema, fornecendo estruturas ou mecânicas fora da partida do xadrez; e (2) "quebras de regra ativas", voltadas a permitir que o jogador modifique, amplie ou distorça as regras durante a própria partida.

7.2.1 Inventário e Posicionamento de Peças

Para que o jogador tenha uma visualização clara das peças de seu exército entre os níveis de dificuldade, foi idealizado e implementado o Inventário. Esse inventário é representado por um tabuleiro extra de 5x5, reutilizando os métodos criados para o tabuleiro principal. Com essa mecânica, o jogador tem a possibilidade de preservar peças entre os níveis de dificuldade, o que adiciona uma camada extra de estratégia. O oponente, por sua vez, não tem acesso a esse inventário, o que significa que ele nunca saberá quais peças estarão disponíveis para o jogador no próximo desafio. Além disso, o jogador, com o inventário, pode posicionar suas peças no início da partida de acordo com a estratégia que preferir, subvertendo em partes o início tradicional do xadrez, uma vez que o jogador só pode posicionar suas peças nas duas primeiras fileiras. Assim, o jogador tem a habilidade de contornar uma dificuldade potencialmente injusta por meio de mecânicas que, na visão do oponente, podem parecer trapaças. No entanto, essas mecânicas são fundamentais para garantir um jogo equilibrado e adaptativo.

Detalhes da Implementação O sistema de inventário foi implementado através do *InventoryController*, que gerencia uma grade secundária de *Tile*s dedicada ao armazenamento de peças do jogador. O controller cria uma matriz de *Tile*s separada do tabuleiro principal, posicionada em uma área dedicada da tela.

O *InventoryController* implementa:

- **Grade de Inventário**: Cria uma grade de *tiles* com *sprites* próprios, diferenciados visualmente dos *tiles* do tabuleiro principal;
- **Gerenciamento de Peças**: Mantém listas de peças disponíveis do jogador (*playerAvailablePieces*) e peças em jogo (*playerPiecesInGame*);
- **Seleção e Movimentação**: Permite seleção e organização de peças dentro do inventário e implementa métodos para movimentação entre inventário e tabuleiro;
- **Posicionamento Controlado**: Controla o posicionamento de peças para as 2 primeiras fileiras do tabuleiro quando permitido.

Modo de Posicionamento de Peças: No início de cada nível, o jogo entra no modo *PlacingPieces*, onde o jogador deve posicionar suas peças do inventário no tabuleiro. Nesse modo, o jogador tem a flexibilidade de mover as peças dentro do tabuleiro e do inventário

livremente, sem ser penalizado por colocar uma peça em um local errado. O *InventoryController* força o posicionamento do rei no tabuleiro, garantindo que o jogador sempre comece com o rei em campo. Conforme ilustrado na Figura 8, durante o modo *PlacingPieces* as *tiles* das duas primeiras fileiras do tabuleiro e as do inventário (à direita) são marcadas com *TileAction.Movement* (realçadas em verde), permitindo posicionamento livre de peças nessas casas; a peça atualmente selecionada é indicada por *TileAction.Selected* (azul).

Figura 8: Modo de posicionamento (*Placing Pieces*), mostrando a alocação inicial de peças do Inventário no tabuleiro.

7.2.2 Loja e Geração de Conjunto de Peças

A aquisição de peças foi estabelecida como uma mecânica essencial antes do início de cada nível, pois, sem ela, o jogador não teria um exército adequado para enfrentar o próximo desafio. Várias abordagens foram inicialmente consideradas: adotar um sistema de moeda tradicional, por exemplo, com recompensas fixas ou baseadas no desempenho (como capturar mais peças ou vencer rapidamente), no entanto essa abordagem tenderia a incentivar estratégias focadas em maximizar o ganho de moedas, o que poderia transformar a estratégia em captura de peças em vez de um confronto estratégico baseado na subversão da desvantagem. Ainda que um sistema de economia pudesse abrir espaço para estratégias de gerenciamento de recursos entre desafios, essa abordagem teria que ser implementada cuidadosamente para evitar que o jogo se torne apenas uma busca por otimizar moedas para progredir. Como alternativa mais alinhada ao propósito do projeto, optou-se por um sistema onde é fornecido diferentes conjuntos de peças a serem escolhidos pelo jogador. Tal sistema se basearia em uma oferta variável e parcialmente aleatória, proporcionando a capacidade da escolha do melhor conjunto para sua estratégia ao reforçar agência do jogador.

Para que essa mecânica fosse expressiva, tornou-se necessário criar conjuntos com perfis diferentes, garantindo que cada escolha representasse, de fato, uma decisão estratégica.

Para manter a imprevisibilidade e a sensação de descoberta, adotou-se a lógica de um sistema de seleção de peças com componentes de aleatoriedade controlada, baseando-se geração procedural. No contexto do jogo, o algoritmo que gera a oferta da loja utiliza hiperparâmetros configuráveis para definir o conjunto de peças disponível em cada nível, garantindo que as escolhas do jogador resultem em combinações distintas de desafios e recursos. Esse processo exemplifica o conceito de *Game Tuning* (refinamento do jogo) [2, Cap. 14: *Games as Emergent Systems*], no qual regras e parâmetros são ajustados, testados e refinados iterativamente para favorecer comportamentos desejáveis em sistemas emergentes. No caso do *Double King*, a oferta de peças, sua frequência, raridade e combinações não são pré-definidas manualmente, mas geradas de forma semi-aleatória a partir de critérios parametrizados de design (como orçamento de dificuldade, variedade e quantidade de peças, além do equilíbrio entre risco e recompensa). Esse modelo permite ajustar a dificuldade, o balanceamento e a diversidade de cada partida sem tornar o jogo previsível ou repetitivo, sustentando uma experiência rica e continuamente envolvente.

Detalhes da Implementação O sistema de Loja foi implementado por meio do componente *SetGenerator* e de controladores associados (*PieceSetSelectorController*, *ShoppingController*). Em conjunto, esses elementos permitem que o jogador adquira novos conjuntos de peças por meio de geração procedural, introduzindo variedade e imprevisibilidade a cada ciclo de jogo.

O método *SetGenerator.GenerateOneSet* implementa um algoritmo de amostragem ponderada sem reposição que consome diretamente os parâmetros dos *ScriptableObjects* configurados (por exemplo, *SetGenerationProfile*, *DifficultyBudgetTable*, *MovementValueTable*, *UpgradeValueTable*). Ele funciona assim:

1. **Cálculo de valor (tabelas)**: para cada tipo selecionado, calcula-se o valor total por unidade combinando tabelas de valor (*PieceValueTable*) [Figura 9]. A quantidade de cada tipo também é sorteada entre mínimo e máximo configuráveis (hiperparâmetros do perfil).

2. **Seleção de tipos (perfil)**: escolhe-se aleatoriamente a quantidade de tipos distintos usando um valor mínimo e máximo configuráveis (hiperparâmetros) no *SetGenerationProfile* [Figura 10]. Os tipos de peça são sorteados por amostragem ponderada usando os pesos do próprio perfil (por exemplo, pesos maiores para Peão aumentam sua probabilidade de aparecer no conjunto).

3. **Ajuste guloso (orçamento e capacidade)**: por fim, aplica-se um ajuste guloso para respeitar as restrições de orçamento e capacidade. O orçamento vem de *DifficultyBudgetTable.GetBudget(difficulty)*;

A configuração do sistema é feita por meio de *ScriptableObjects* que permitem o balanceamento sem necessidade de alterar código. Perfis de geração podem ser criados para privilegiar diferentes características: maior quantidade de peças baratas, menor quantidade

de peças caras ou configurações balanceadas. O orçamento vinculado à dificuldade influencia diretamente as ofertas geradas, incorporando o nível de desafio ao próprio processo de geração procedural. Foram criadas quatro opções de escolha de conjunto de peças, cada uma baseada em perfis distintos de geração. A primeira utiliza um perfil focado em maior quantidade de peças, resultando naturalmente em valores individuais menores; a segunda adota um perfil centrado em alto valor, priorizando peças mais valiosas em menor quantidade; e as duas últimas derivam de um mesmo perfil balanceado, combinando peças de valor intermediário em quantidades mais amplas.

```
[Header("Base values per piece type")]
public int pawn = 1, knight = 3, bishop = 3, rook = 5, queen = 9;
```

Figura 9: Valores base e probabilidades em tabelas (*Piece/Movement/Upgrade Value Tables*).

```
[Header("Distinct piece types per set (1..3)")]
[Range(1, 3)] public int minDistinctTypes = 1;
[Range(1, 3)] public int maxDistinctTypes = 3;

[Header("Quantity per selected type (min..max)")]
[Min(1)] public int minQuantityPerType = 1;
[Min(1)] public int maxQuantityPerType = 4;

[Header("Piece type weights (relative likelihood)")]
public int pawnWeight = 10;
public int knightWeight = 6;
public int bishopWeight = 6;
public int rookWeight = 3;
public int queenWeight = 1;
```

Figura 10: Hiperparâmetros do perfil de geração (*SetGenerationProfile*).

Com essa mecânica, o jogo passou a dispor, em termos estruturais, de início, meio e fim: o jogador tem condições de construir um exército gradulamente e, por meio de decisões estratégicas, concluir os desafios. Ainda assim, os testes indicaram que a dificuldade permanecia elevada. Em determinados níveis, o conjunto gerado não era suficiente para lidar com o exército do oponente. Esse cenário exigiu novos ciclos de *tuning* dos hiperparâmetros, resultando em ajustes no orçamento de dificuldade e na quantidade de peças oferecidas nos níveis iniciais, de forma a manter o conflito intenso, porém vencível.

7.3 Quebrando as Regras Ativamente

Com as mecânicas estruturais estabelecidas na etapa anterior, tornou-se possível explorar uma segunda modalidade de subversão das regras tradicionais do xadrez: aquelas que atuam durante a própria partida. Essa idealização e prototipação se desdobrou em duas direções:

1. mecânicas que atuam sobre as peças — ampliando seus padrões de movimento, modificando sua persistência ou criando novas formas de interação — e que puderam ser efetivamente implementadas;

2. mecânicas voltadas à alteração de regras estruturais do sistema, como reorganização do tabuleiro, modificação de turnos ou ajustes em outras mecânicas. Essas propostas foram concebidas conceitualmente, mas permaneceram no plano de design dentro do escopo deste projeto, servindo como base para investigações futuras.

7.3.1 Peças Evoluídas

A partir da arquitetura base e expansível desenvolvida, tornou-se possível tratar as peças como entidades capazes de evolução. A implementação das Peças e *engine* do oponente artifical já previa a capacidade de acoplar novos padrões de movimento e de integrá-los ao jogo por meio de variantes do xadrez. Somado a isso, tornou-se viável criar atributos específicos para as peças, permitindo que, em determinadas situações, a validação dessa habilidade segueria um fluxo que substituí ou complementa regras tradicionais do xadrez. Essa abordagem abriu espaço para um sistema de habilidades que dialoga diretamente com a lógica emergente do jogo. Assim, foi possível intensificar a experiência de subversão das regras clássicas, preservando sua estrutura essencial, mas introduzindo exceções controladas que ampliam o espaço de possibilidade disponível ao jogador.

Detalhes da Implementação Do ponto de vista de implementação, a adição de novos movimentos aproveita diretamente a infraestrutura já compatível com o *FairyStockfish*, permitindo, por exemplo, combinar os padrões de movimentação do cavalo e do bispo em uma única peça. Essa compatibilidade foi importante, tendo em vista que sem ela, o oponente artificial não teria consciência de que determinadas peças possuem movimentações híbridas, podendo mover o rei para casas atacadas, ocasionando na eventual captura do Rei (mecânica que poderia ser explorada no futuro de forma mais balanceada). A escolha do *Fairy-Stockfish*, portanto, garantiu que o adversário artificial entendesse as regras expandidas, ainda que não possua em seu arsenal tais peças modificadas.

A introdução dessa funcionalidade revelou a necessidade de expandir o *SetGeneration-Profile* com novos hiperparâmetros, permitindo que peças com padrões híbridos também fossem disponibilizadas na Loja. Assim, a decisão de adquirir ou não peças com movimentos adicionais passou a integrar o espaço estratégico do jogador, enriquecendo a construção do exército e criando novas linhas de planejamento.

Além da expansão dos movimentos, foi idealizado um sistema complementar de habilidades que alteram regras fundamentais do comportamento das peças — desde o que ocorre quando capturam outra peça, até o que acontece quando são capturadas. Como

consequência da existência do inventário, surgiram naturalmente habilidades voltadas à interação entre tabuleiro e armazenamento. Para organizar essa lógica, três categorias iniciais foram definidas e, por facilitar a comunicação visual, denominadas R, G e B, como ilustrado na Tabela 1.

Canal	**Habilidade**
R	Quando o jogador captura uma peça do oponente, essa peça é adicionada ao inventário do jogador.
G	Quando a peça do jogador é capturada, ela retorna ao inventário em vez de ser destruída.
B	Permite remover essa peça do tabuleiro e enviá-la de volta ao inventário.

Tabela 1: Habilidades dos canais RGB

A metáfora da luz serviu de base conceitual: cada habilidade é representada por uma cor, onde as combinações de habilidades são exibidas por meio da mistura dessas cores (ver Seção 8.1). Isso permite que o jogador identifique imediatamente, pela tonalidade final, quais habilidades uma peça possui, incluindo habilidades híbridas derivadas da união de duas ou três cores, caso novas expansões sejam exploradas futuramente.

Aproveitando esse conceito, caso a peça possua simultaneamente as três habilidades (R, G e B), foi implementada uma quarta habilidade adicional: a capacidade de retornar do inventário diretamente ao tabuleiro, introduzindo um elemento de surpresa e reforçando o caráter de transgressão sancionada.

O sistema RGB foi implementado por meio da estrutura *PiecesUpgrades*, na qual cada habilidade é representada por uma *flag* booleana. A lógica de ativação consiste em verificar a presença da habilidade antes da execução da regra padrão, desviando o fluxo para comportamentos alternativos quando aplicável. Isso pode ser observado na Figura 11, que exemplifica o método de captura: se a peça capturada possui a habilidade G, um fluxo alternativo é acionado; se possui a habilidade R, um outro fluxo alternativo é acionado; caso contrário, segue-se o comportamento tradicional, removendo o objeto do jogo.

```
public void CapturePiece(Tile fromTile, Tile toTile)
{
    if (toTile != null && toTile.piece != null)
    {
        Piece attacker = fromTile.piece;
        Piece captured = toTile.piece;

        if (captured.HasUpgradeG())
        {
            pieceController.HandleCapturedUpgradeG(captured, toTile);
        }
        else if (attacker.HasUpgradeR())
        {
            pieceController.HandleAttackerUpgradeR(attacker, captured, toTile);
        }
        else
        {
            pieceController.DestroyPiece(toTile.piece);
            toTile.RemovePiece();
        }
    }
}
```

Figura 11: Fluxo de captura: desvio para habilidades RGB e caminho padrão (*BoardController.CapturePiece*).

Durante os *playtests*, essa mecânica se desdobrou em cenários particularmente interessantes. A habilidade R, por exemplo, permite ao jogador "roubar" peças do adversário, compensando a raridade de determinadas unidades na Loja. Combinações avançadas, como peças que reúnem R, G e B, aproximam-se de uma quebra quase absoluta das regras: tais peças podem capturar, retornar ao inventário quando destruídas e voltar ao tabuleiro posteriormente, funcionando como uma espécie de "soldado imortal" dentro da narrativa de exércitos do xadrez.

Para evitar que esse comportamento se tornasse dominante, os hiperparâmetros associados às habilidades RGB foram ajustados para torná-los raros nos perfis da Loja, preservando seu caráter de recompensa excepcional. Além disso, movimentos entre inventário e tabuleiro contam como jogadas, o que balanceia seu uso em dificuldades mais altas, tendo em vista que o oponente reage ao surgimento dessa peça antes dela poder ser mover, podendo cravar ela ou prosseguir para outras estratégias tradicionais do xadrez.

7.3.2 Mecânicas Idealizadas: Cartas de Alteração de Regras

A partir da base construída surgiu a idealização de um sistema de cartas dedicado à modificação explícita de regras. Inspirado por jogos do gênero *deckbuilder*, como o próprio *Balatro*, estilo que permite que cartas funcionem como elementos capazes de alterar condições do jogo, ampliando ainda mais o espectro de transgressão sancionada já existente no *Double*

King.

Embora não tenham sido implementadas nesta versão do protótipo, essas ideias constituem um produto do processo de game design desenvolvido. Elas resultam diretamente das mecânicas concebidas e da base teórica estabelecida até então. As cartas, idealizadas para manipular regras alternativas, foram esboçadas como ferramentas capazes de introduzir novas exceções, modificar princípios clássicos do xadrez ou expandir mecânicas já subversivas. Entre as propostas concebidas, destacam-se cartas que poderiam adicionar ou remover fileiras ou colunas, alterando o princípio do tabuleiro quadrado, permitir que uma peça se mova duas vezes seguidas ou, ainda, funcionar como um último recurso capaz de ativar o modo *Double King* após um xeque-mate. Assim, mesmo permanecendo em nível conceitual, esse conjunto de propostas consolida direções de expansão futuras e demonstra como o sistema pode continuar evoluindo de maneira emergente, modular e experimental.

8 Experiência do Usuário

A experiência do usuário foi estruturada em dois eixos complementares: a identidade visual, responsável por comunicar a atmosfera e a narrativa do projeto, e a interface do usuário, responsável por tornar legíveis as regras, os estados do sistema e as ações possíveis, que deixam de ser apenas elementos estéticos e tornam-se fundamentais para garantir *Meaningful Play*.

Segundo Salen e Zimmerman (2003), o jogo só produz significado quando as ações do jogador são discerníveis, ou seja, quando o jogador compreende claramente o que sua ação causou, e integrados, quando o resultado dessa ação se conecta ao restante do sistema, influenciando situações posteriores. No *Double King*, no qual habilidades extras, regras alternativas e comportamentos excepcionais se acumulam sobre a estrutura do xadrez clássico, a clareza visual e a legibilidade tornam-se indispensáveis para que essas ações sejam perceptíveis e integradas ao fluxo estratégico do jogo.

8.1 Identidade Visual

A identidade visual foi desenvolvida com o objetivo de reforçar a atmosfera do projeto, uma releitura do xadrez marcada por conflito, experimentação e subversão progressiva das regras tradicionais. Para isso, optou-se pela criação de artes originais em *pixel art*, produzidas no *LibreSprite*. A escolha desse estilo não foi apenas estética, mas funcional, a leitura clara de peças, *tiles*, efeitos e estados é crucial para manter a compreensão necessária para o *Meaningful Play*.

O conjunto de *sprites* criado reflete essa preocupação. As peças foram representadas tanto em sua versão tradicional quanto em versões evoluídas, nas quais combinações RGB comunicam visualmente habilidades ativas, permitindo que o jogador identifique, de imediato, a presença de efeitos especiais [Figura 13]. As *tiles* do tabuleiro também receberam variações visuais, diferenciando áreas de movimento, casas atacadas, casas válidas, tiles removíveis e tiles especiais do modo *Double King*, reforçando a leitura espacial necessária para ações estratégicas integradas (como ilustrado nas figuras apresentadas ao longo do relatório).

Figura 12: Exemplo de peças com diferentes combinações de upgrades RGB.

A arte dos chefes (*bosses*) foi elaborada para associar cada nível de dificuldade a uma entidade visual própria, fortalecendo a sensação de progressão dentro da narrativa. Para fins de ilustração, cada figura apresentada ao longo do relatório utilizou um chefe diferente, de modo a exemplificar visualmente os distintos níveis e desafios do jogo. A tela inicial, apresentada na Figura 13, busca transmitir a essência do *Double King*: um confronto direto entre dois reis, o Rei Branco e o Rei Preto.

Figura 13: Tela inicial (*Main Menu*) do jogo.

Essa construção estética garantiu uma identidade visual coesa, capaz de comunicar rapidamente estados, funções e identidades de cada elemento. Ao mesmo tempo, ela sustenta os dois pilares do *Meaningful Play*: cada ação permanece perceptível e compreensível, e suas consequências continuam conectadas ao desenvolvimento da partida como um todo.

8.2 Interface do Usuário

Se a identidade visual estabelece o caráter estético e temático do *Double King*, é a interface do usuário (UI) que torna possível ao jogador compreender, de forma clara e contínua, o que está acontecendo no sistema.

A UI foi construída para comunicar, de modo imediato, em que estado o jogo se encontra e quais ações são possíveis naquele momento. A Figura 14 ilustra esse princípio: ao selecionar uma peça, a interface destaca visualmente sua posição, exibe seus movimentos válidos no tabuleiro e apresenta, no painel lateral, suas informações essenciais: tipo, habilidades RGB e padrões de movimento. Esse mesmo painel informa qual jogador deve agir, qual é o nível de dificuldade, qual estado do sistema está ativo e quais instruções orientam o próximo passo (“Selecione uma peça”, “Aguarde o oponente”, “Remova uma *tile*” etc.). Essa combinação de *feedback* imediato, instrução contextual e sinalização espacial é fundamental para manter a experiência legível mesmo quando o comportamento das peças se afasta das regras tradicionais do xadrez.

Figura 14: Elementos de interface relacionados ao estado do jogo.

9 Resultados

Após a construção da identidade visual, da interface e das diversas mecânicas que compõem o protótipo, torna-se possível examinar de forma integrada aquilo que o projeto efetivamente produz como experiência e como sistema. Os resultados obtidos demonstram que o *Double King* cumpriu de maneira consistente os objetivos estabelecidos, articulando fundamentação teórica, escolhas de design e implementação técnica em um protótipo funcional e coerente com o propósito investigativo do trabalho.

Partindo de um sistema formal estável, o xadrez, o projeto evoluiu para um ecossistema de mecânicas que introduzem incerteza, emergência e transgressão sancionada, ampliando o espaço de possibilidade do jogador sem comprometer a legibilidade das ações. Esse percurso materializa, na prática, os princípios de regras dinâmicas, sistemas emergentes e *meaningful play* que orientaram o trabalho desde sua concepção, permitindo observar como decisões de design e ciclos de prototipação se combinam para gerar experiências de jogo significativamente distintas do sistema de origem.

Ainda que a ideia inicial de "regras evolutivas e mecânicas customizáveis" estivesse prevista nos objetivos gerais, a forma específica que essas mecânicas assumiram não foi definida previamente, ela emergiu diretamente do processo iterativo de design. As soluções implementadas (inventário, loja com geração procedural, peças evoluídas, modo *Double King*, sistema de posicionamento e habilidades RGB) surgiram como respostas a problemas concretos identificados durante a prototipação, tais como a injustiça estrutural do conflito, a necessidade de progressão significativa e a manutenção da rejogabilidade. Cada decisão de design foi sustentada por referenciais teóricos, validada por experimentação e ajustada por ciclos sucessivos de teste, em consonância com a metodologia *playcentric* de Fullerton

(2024), na qual teoria e prática evoluem em diálogo permanente.

Do ponto de vista formativo, o protótipo também cumpre o papel de síntese integradora das competências desenvolvidas ao longo da graduação. A arquitetura modular construída em *Unity*, a integração com o motor *Fairy-Stockfish*, a implementação de algoritmos de geração procedural, a modelagem de sistemas, a criação de arte 2D original, a análise de referências e a documentação crítica constituem um conjunto de resultados que evidenciam a consolidação de uma base técnica sólida.

9.1 Análise Conceitual do Jogo

O arcabouço conceitual *Rules, Play e Culture* permite observar diferentes dimensões do jogo: sua estrutura formal, sua experiência e seu contexto. Ao examinar o *Double King* a partir dessas lentes, torna-se possível compreender não apenas como as mecânicas funcionam, mas como elas se articulam para produzir uma experiência de jogo significativa, já que cada uma dessas dimensões ilumina aspectos fundamentais da relação entre ação, consequência e interpretação dentro do jogo.

9.1.1 Esquemas Formais (RULES)

Do ponto de vista formal, o *Double King* caracteriza-se como um sistema de incerteza e emergência. Em vez de um conjunto estável e determinístico de situações — como ocorre no xadrez clássico, um jogo de informação perfeita —, o protótipo introduz múltiplas fontes de variabilidade, que isoladamente, parecem regras simples, mas em conjunto, produzem comportamentos que não podem ser totalmente antecipados [2, Cap. 14: *Games as Emergent Systems*].

A incerteza se manifesta tanto em nível *micro* (quais peças serão oferecidas na loja, quais habilidades estarão disponíveis) quanto em nível *macro* (quais configurações de exército emergirão ao longo de uma partida) [2, Cap. 15: *Games as Systems of Uncertainty*]. Ao transformar um jogo de informação perfeita em um sistema de informação imperfeita, o *Double King* situa o jogador em um espaço contínuo de risco e experimentação.

Nesse contexto, a transgressão sancionada torna-se um elemento central, permitindo que o jogador quebre, de maneira controlada, a lógica tradicional do xadrez [2, Cap.21 : *Breaking the Rules*]. A assimetria extrema criada pelo exército estável do oponente torna-se interessante porque o jogador é incentivado a usar essas quebras de regra para superar uma injustiça estrutural.

9.1.2 Esquemas Experienciais (PLAY)

A experiência construída pelo *Double King* articula uma combinação de cálculo tático, descoberta e adaptação contínua que dialoga diretamente com o campo da experiência no jogo [2, Cap. 23 : *Play as the Game of Experience*]. As decisões sobre qual peça adquirir, como organizar o inventário, quando aceitar um empate e de que forma explorar habilidades especiais fazem com que o jogador participe ativamente da construção do próprio percurso. Cada partida passa a ser vivida como uma sequência encadeada de acontecimentos, na qual escolhas iniciais influenciam estados posteriores de forma perceptível.

Os ciclos de risco, frustração e recompensa presentes no jogo também dialogam com a perspectiva do prazer no ato de jogar [2,Cap. 24: *Games as the Play of Pleasure*]. Enfrentar um conflito estruturalmente desfavorável, sobreviver a turnos críticos, recuperar peças por meio de habilidades especiais e descobrir combinações eficazes produz um ritmo emocional característico de jogos que engajam pelo desafio. A alternância entre tensão e alívio, risco e compensação, contribui diretamente para que a experiência permaneça motivadora mesmo diante das incertezas impostas pelo sistema.

9.1.3 Esquemas Contextuais (CULTURE)

Como afirmam Salen e Zimmerman (2003) , "criar jogos é também criar cultura": todo jogo, ao ser projetado, inevitavelmente expressa valores, hierarquias e modos de interpretar o mundo [2, Cap.30: *Games as Cultural Rhetoric*]. Sob essa perspectiva, o *Double King* evidencia como escolhas de design podem reconfigurar estruturas culturais já consolidadas. Ao tomar o xadrez como ponto de partida, um sistema formalmente rígido e amplamente reconhecido, e ao expandi-lo com mecânicas que flexibilizam sua lógica, o protótipo opera no espaço cultural entre tradição e transformação.

As mecânicas introduzidas deslocam o sistema original em direção a retóricas culturais associadas à criatividade, agência e resistência. O jogador é colocado diante de um sistema desigual e convidado a intervir nele, assumindo um papel ativo na reconstrução das possibilidades do jogo. Essa abordagem aproxima o projeto de diferentes retóricas , entre elas a retórica do poder (herdada do xadrez), a retórica da imaginação (nas combinações híbridas de movimentos), e, sobretudo, a retórica da frivolidade, que permite questionar e subverter regras estabelecidas. Assim, mais do que refletir valores culturais existentes, o *Double King* cria um espaço em que a contestação lúdica das estruturas rígidas torna-se parte significativa da experiência.

9.2 Playtesting

Apesar de o protótipo se alinhar conceitualmente às estruturas que sustentam o meaningful play, foi o processo de teste, e não apenas a teoria, que determinou, de fato, a qualidade da experiência obtida. Ao longo da prototipação, ciclos contínuos de *playtesting* revelaram problemas de clareza, balanço e funcionalidade que não poderiam ser antecipados apenas pela análise teórica, permitindo ajustar dificuldades, revisar fluxos, eliminar ambiguidades e aprimorar a legibilidade das ações. Esse movimento reflete exatamente o que a literatura descreve sobre o design iterativo: somente jogando e observando o jogo em ação é possível compreender como suas regras se traduzem em experiência [2, Cap. 2: *The Design Process*]. Os *feedbacks* coletados guiaram decisões centrais de melhoria, sustentando o equilíbrio entre risco, agência e descoberta. Ao final desse processo, o *Double King* apresentou estrutura suficiente para produzir uma experiência significativa, pela teoria e prática, mas que ainda pode ser expandido e melhorado para despertar ainda mais a experiência significativa durante o jogo .

10 Conclusões e Trabalhos Futuros

O projeto contribui de forma teórica e prática para o campo de game design ao articular, em um protótipo funcional, conceitos fundamentais discutidos na literatura. A partir da construção de um sistema inspirado no xadrez, mas estruturado em torno de regras evolutivas e mecânicas customizáveis, foi possível explorar, na prática, ideias sobre sistemas de conflito, incerteza, emergência, transgressão sancionada e *meaningful play*.

Em termos conceituais, o trabalho demonstra como o uso de um arcabouço teórico sólido pode orientar decisões de design desde o nível micro (ajuste de algoritmos, orçamentos de dificuldade, configuração de peças) até o nível macro (estrutura de progressão, identidade do jogo, papel do jogador no sistema). Em termos práticos, o projeto resultou em um protótipo jogável que integra um oponente aritifical baseada em xadrez, geração procedural de recursos, mecânicas de inventário, modos de jogo alternativos, regras dinâmicas e uma identidade visual consistente.

Do ponto de vista da formação em Engenharia da Computação, o *Double King* funciona como síntese de competências desenvolvidas ao longo do curso: modelagem de sistemas, programação orientada a objetos, integração com bibliotecas externas, desenvolvimento iterativo e documentação técnica. O processo de desenvolvimento também evidencia a importância do *playtesting* como ferramenta de investigação empírica, permitindo testar hipóteses de design e ajustar parâmetros em função do comportamento observado.

Embora o protótipo desenvolvido seja plenamente suficiente para sustentar a análise teórica proposta e demonstrar, na prática, a criação de *meaningful play*, o sistema construído abre espaço para expansões que poderiam transformar o *Double King* em um jogo completo e potencialmente publicável. A base técnica, estética e conceitual estabelecida ao longo do projeto permite vislumbrar caminhos de aprofundamento que ampliariam a complexidade estratégica, a legibilidade e a riqueza do jogo. Entre essas possibilidades, destacam-se:

- aprofundar o sistema de cartas de alteração de regras, integrando-o de forma plena ao ciclo de jogo e reforçando a mecânica de quebra de regras;
- explorar modos adicionais de dificuldade com maior variação procedural na composição dos exércitos inimigos, fortalecendo a rejogabilidade e o caráter *roguelike* do projeto;
- refinar a interface e o feedback visual para comunicar com ainda mais clareza habilidades, estados especiais e consequências das ações, aprimorando a discernibilidade;
- conduzir estudos de *playtesting* com diferentes perfis de jogadores, investigando percepções de dificuldade, justiça e, sobretudo, *meaningful play*.

Essas direções apontam para a continuidade natural do projeto, indicando como o protótipo pode evoluir em direção a um artefato mais completo, aprofundando seu potencial expressivo, sistêmico e cultural.

Em síntese, o *Double King* não se encerra como produto final, mas como um ponto de partida robusto para investigações futuras sobre como jogos podem usar sistemas formais clássicos, como o xadrez, como base para experimentação com regras evolutivas e mecânicas customizáveis, e por meio da emergência poder produzir *meaningful play*.

Referências

[1] Shannon, Claude E. "XXII. Programming a computer for playing chess." *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, vol. 41, no. 314, 1950.

[2] Katie Salen Tekinbas and Eric Zimmerman. *Rules of Play: Game Design Fundamentals*. MIT Press, 2003.

[3] Thompson, J. Mark. "Defining the abstract." *Game & Puzzle Design*, vol. 1, no. 1, 2015.

[4] Szabados, György, et al. "Roguelike games: The way we play." 2022.

[5] Unity Documentation. Disponível em: `https://docs.unity3d.com/Manual/index.html` (Acesso em: 21 nov. 2025).

[6] Fullerton, Tracy. *Game Design Workshop: A Playcentric Approach to Creating Innovative Games*. AK Peters/CRC Press, 2024.

[7] LocalThunk. A Timeline do Balatro. LocalThunk, [s.d.]. Disponível em: `https://localthunk.com/blog/balatro-timeline-3aarh`. Acesso em: out. 2025.

[8] Fairy-Stockfish. Get involved. Fairy-Stockfish, [s.d.]. Disponível em: `https://fairy-stockfish.github.io/get-involved/`. Acesso em: 23 set. 2025.

[9] Yu, H.; Moldenhauer, C.; Moldenhauer, J. *The Art of Cuphead*. Milwaukee: Dark Horse Books, 2020.

[10] LibreSprite. LibreSprite. LibreSprite, [s.d.]. Disponível em: `https://libresprite.github.io/`. Acesso em: ago. 2025.

A Apêndice

A.1 Algoritmo de Escolha com Pesos do Double King Mode

O algoritmo *ChooseWeightedByDistance* implementa a escolha probabilística de *tiles* no Double King Mode, calculando pesos baseados na distância Manhattan ao rei adversário:

```
private Tile ChooseWeightedByDistance(List<Tile> candidates,
    Vector2Int reference, bool preferCloser,
    float preferenceSharpness = 1f)
{
    if (candidates == null || candidates.Count == 0) return null;

    float total = 0f;
    float[] weights = new float[candidates.Count];
    for (int i = 0; i < candidates.Count; i++)
    {
```

```
        var pos = candidates[i].position;
        int d = Mathf.Abs(pos.x - reference.x) +
                Mathf.Abs(pos.y - reference.y);
        float baseW = preferCloser ? (1f / (1f + d)) : (1f + d);
        float w = preferenceSharpness != 1f ?
            Mathf.Pow(baseW, preferenceSharpness) : baseW;
        weights[i] = w;
        total += w;
    }

    int chosenIndex = ChooseIndexByWeights(weights, total);
    return candidates[chosenIndex];
}

private int ChooseIndexByWeights(float[] weights, float total)
{
    if (weights == null || weights.Length == 0) return 0;
    if (total <= 0f)
    {
        return Mathf.FloorToInt(Random.value * weights.Length) % weights.Length;
    }

    float r = Random.value * total;
    for (int i = 0; i < weights.Length; i++)
    {
        if (r < weights[i]) return i;
        r -= weights[i];
    }
    return weights.Length - 1;
}
```