

INSTITUTO FEDERAL DO TOCANTINS - CAMPUS COLINAS DO TOCANTINS
TECNÓLOGO EM ANÁLISE E DESENVOLVIMENTO DE SISTEMAS
DANILO COSTA MATIAS

DESENVOLVIMENTO DE UM SIMULADOR DE XADREZ COM INTEGRAÇÃO
DO MOTOR STOCKFISH UTILIZANDO A GODOT

COLINAS DO TOCANTINS - TO
2026

DANILO COSTA MATIAS

**DESENVOLVIMENTO DE UM SIMULADOR DE XADREZ COM INTEGRAÇÃO
DO MOTOR STOCKFISH UTILIZANDO A GODOT**

Trabalho de curso apresentado ao curso de Tecnologia do Instituto Federal do Tocantins – Campus Colinas do Tocantins, como requisito parcial para a obtenção do título de Tecnólogo em Análise e Desenvolvimento de Sistemas, sob orientação do Prof. Me. Fernando Turibio de Moura.

COLINAS DO TOCANTINS - TO

2026

Dados Internacionais de Catalogação na Publicação (CIP)
Bibliotecas do Instituto Federal do Tocantins

M433d Matias, Danilo Costa

Desenvolvimento de um simulador de xadrez com integração do motor stockfish utilizando a Godot / Danilo Costa Matias. – Colinas do Tocantins, TO, 2026.

47 f. : il. color.

Trabalho de Conclusão de Curso (Curso Superior de Tecnologia em Análise e Desenvolvimento de Sistemas) – Instituto Federal de Educação, Ciência e Tecnologia do Tocantins, Campus Colinas do Tocantins, Colinas do Tocantins, TO, 2026.

Orientador: Me. Fernando Turibio de Moura

1. Simulador de Xadrez. 2. Motor Stockfish. 3. Godot engine. I. Turibio de Moura, Fernando. II. Título.

CDD 004.2

A reprodução total ou parcial, de qualquer forma ou por qualquer meio, deste documento é autorizada para fins de estudo e pesquisa, desde que citada a fonte.

Elaborado pelo sistema de geração automática de ficha catalográfica do IFTO com os dados fornecidos pelo(a) autor(a).

DANILO COSTA MATIAS

DESENVOLVIMENTO DE UM SIMULADOR DE XADREZ COM INTEGRAÇÃO DO MOTOR STOCKFISH UTILIZANDO A GODOT

Trabalho de curso apresentado ao curso de Tecnologia do Instituto Federal do Tocantins – Campus Colinas do Tocantins, como requisito parcial para a obtenção do título de Tecnólogo em Análise e Desenvolvimento de Sistemas, sob orientação do Prof. Me. Fernando Turibio de Moura.

Banca Examinadora

Documento assinado digitalmente
FERNANDO TURIBIO DE MOURA
Data: 12/06/2026 16:53:06-0300
Verifique em <https://validar.iti.gov.br>

Me. Fernando Turibio de Moura
Instituto Federal do Tocantins Campus Colinas do Tocantins

Documento assinado digitalmente
RONNEESLEY MOURA TELES
Data: 11/06/2026 10:06:09-0300
Verifique em <https://validar.iti.gov.br>

Dr. Ronneesley Moura Teles
Instituto Federal do Tocantins Campus Colinas do Tocantins

Documento assinado digitalmente
MATEUS NUNES DOS SANTOS
Data: 11/06/2026 18:01:24-0300
Verifique em <https://validar.iti.gov.br>

Esp. Mateus Nunes dos Santos
Instituto Federal do Tocantins Campus Colinas do Tocantins

Aprovada em 29 de maio de 2026

Dedico este trabalho à minha esposa, pelo apoio incondicional, e a todos que acreditaram no meu potencial mesmo nos momentos mais desafiadores desta jornada.

AGRADECIMENTOS

Primeiramente, a Deus, pela força, resiliência e clareza mental concedidas ao longo de toda esta jornada, permitindo-me superar os momentos de cansaço e os desafios inerentes à pesquisa e ao desenvolvimento de software.

Ao Instituto Federal de Educação, Ciência e Tecnologia do Tocantins (IFTO) - Campus Colinas do Tocantins, por ser o alicerce da minha formação profissional. Aos excelentes professores do curso de Tecnologia em Análise e Desenvolvimento de Sistemas (TADS), que não apenas compartilharam seus conhecimentos teóricos e práticos, mas também me inspiraram a buscar sempre a excelência e a inovação tecnológica.

Um agradecimento muito especial ao meu orientador, Prof. Me. Fernando Turibio de Moura. Sua paciência, dedicação e vasto conhecimento foram fundamentais para transformar um projeto de alta complexidade técnica em um sistema real. Obrigado pela confiança no meu potencial, pelas revisões minuciosas e por me guiar com maestria pelas melhores práticas de engenharia de software, especialmente nos momentos mais complexos de integração deste simulador.

À minha esposa e à minha família, pela compreensão inesgotável nas inúmeras noites, madrugadas e finais de semana dedicados à programação, à resolução de falhas lógicas e à escrita deste documento. Sem o suporte emocional, o incentivo constante e a paciência de vocês com as minhas ausências, a conclusão desta etapa não seria possível. Vocês são a minha maior motivação.

À vasta comunidade de desenvolvedores de software livre, em especial aos criadores e mantenedores da Godot Engine, da linguagem Python e do motor de xadrez Stockfish. A filosofia de código aberto, a documentação acessível e o compartilhamento global de conhecimento foram a base técnica que viabilizou a materialização e o sucesso deste sistema.

Aos meus colegas de turma e amigos de curso, por todas as linhas de código revisadas em conjunto, pelos testes incansáveis do jogo em suas diversas versões, pelas partidas jogadas e pelas horas de estudo compartilhadas. Vocês tornaram os desafios desta graduação muito mais leves e a jornada muito mais inesquecível.

Por fim, a todos que, de forma direta ou indireta, contribuíram com palavras de incentivo, críticas construtivas e apoio ao longo destes anos de graduação, meu mais sincero muito obrigado.

Quero que os jogadores experimentem coisas que só são possíveis dentro dos videogames e que nunca foram feitas antes. Do contrário, não há sentido em criar o jogo.

Hideo Kojima

RESUMO

Este trabalho apresenta o desenvolvimento de um simulador de xadrez com integração ao motor de inteligência artificial (IA) Stockfish, utilizando a Godot Engine. O objetivo principal consistiu em construir uma aplicação funcional, customizável e com alto nível competitivo, solucionando o desafio técnico de integrar uma inteligência artificial complexa a um motor de jogos moderno. A metodologia adotada envolveu a criação da interface gráfica e da lógica de validação de regras, como o roque e o *en passant*, utilizando a linguagem nativa GDScript. Para o processamento das jogadas do oponente virtual, utilizou-se a linguagem Python como *middleware*, responsável por estabelecer a comunicação assíncrona entre o frontend na Godot e o motor Stockfish, operando através do protocolo *Universal Chess Interface* (UCI). Adicionalmente, foi implementado um sistema dinâmico de temas visuais baseado em dicionários e matrizes de dados, permitindo a personalização da interface gráfica em tempo real. Os resultados obtidos demonstram que a arquitetura proposta garante fluidez na jogabilidade sem comprometer o desempenho computacional durante o cálculo das melhores jogadas. Conclui-se que a integração entre Godot, Python e Stockfish forma uma base robusta, eficiente e altamente modular para o desenvolvimento de jogos de tabuleiro digitais de alta complexidade técnica.

Palavras-chave: Xadrez. Godot Engine. Python. Inteligência Artificial. Stockfish.

ABSTRACT

This work presents the development of a chess simulator integrated with the Stockfish artificial intelligence (AI) engine, using the Godot Engine. The main objective was to build a functional, customizable, and highly competitive application, solving the technical challenge of integrating a complex artificial intelligence into a modern game engine. The adopted methodology involved creating the graphical interface and the rule validation logic, such as castling and *en passant*, using the native GDScript language. To process the virtual opponent's moves, the Python language was used as *middleware*, responsible for establishing asynchronous communication between the Godot frontend and the Stockfish engine, operating through the *Universal Chess Interface* (UCI) protocol. Additionally, a dynamic visual theme system based on dictionaries and data arrays was implemented, allowing for real-time customization of the graphical interface. The results obtained demonstrate that the proposed architecture ensures fluid gameplay without compromising computational performance during the calculation of the best moves. It is concluded that the integration between Godot, Python, and Stockfish forms a robust, efficient, and highly modular foundation for the development of digital board games of high technical complexity.

Keywords: Chess. Godot Engine. Python. Artificial Intelligence. Stockfish.

LISTA DE ILUSTRAÇÕES

Figura 1 – Página do repositório <i>Open-Source</i>	22
Figura 2 – Visualização do canal no YouTube	22
Figura 3 – Ambiente de partida contra o computador na plataforma Lichess	23
Figura 4 – Layout do software Lucas Chess	23
Figura 5 – Visual do site Chess.com	24
Figura 6 – Diagrama de Implantação da estrutura técnica do Royal Legacy	29
Figura 7 – Layout do menu principal	30
Figura 8 – Página de configurações de áudio, temas e créditos	30
Figura 9 – Ambiente final do tabuleiro com peças <i>pixel art</i> e botões de controle	31
Figura 10 – Diagrama Entidade Relacionamento para persistência de preferências locais	31
Figura 11 – Diagrama de casos de uso do simulador	32
Figura 12 – Diagrama de classes detalhando a estrutura em GDScript e Python	33
Figura 13 – Ambiente final de seleção de temas visuais	37
Figura 14 – Painel final de controle de canais de áudio	38
Figura 15 – Ambiente de partida com aplicação de temas e realce de movimentos	39

LISTA DE TABELAS

Tabela 1 – Métricas de tempo de resposta da IA Stockfish e estabilidade da renderização 39

LISTA DE ABREVIATURAS E SIGLAS

API	<i>Application Programming Interface</i> (Interface de Programação de Aplicações)
CPU	<i>Central Processing Unit</i> (Unidade Central de Processamento)
DER	Diagrama Entidade-Relacionamento
DFD	Diagrama de Fluxo de Dados
FEN	<i>Forsyth-Edwards Notation</i> (Notação Forsyth-Edwards)
FIDE	Federação Internacional de Xadrez (<i>Fédération Internationale des Échecs</i>)
FPS	<i>Frames Per Second</i> (Quadros por Segundo)
GDScript	<i>Godot Script</i> (Linguagem de <i>script</i> nativa da Godot)
GUI	<i>Graphical User Interface</i> (Interface Gráfica do Usuário)
HD	<i>Hard Disk</i> (Disco Rígido)
IA	Inteligência Artificial
AI	<i>Artificial Intelligence</i> (Inteligência Artificial)
IDE	<i>Integrated Development Environment</i> (Ambiente de Desenvolvimento Integrado)
IFTO	Instituto Federal de Educação, Ciência e Tecnologia do Tocantins
JSON	<i>JavaScript Object Notation</i> (Notação de Objetos JavaScript)
NNUE	<i>Efficiently Updatable Neural Network</i> (Rede Neural Eficientemente Atualizável)
PGN	<i>Portable Game Notation</i> (Notação Portátil de Jogo)
PvE	<i>Player vs Environment</i> (Jogador contra Máquina)
PvP	<i>Player vs Player</i> (Jogador contra Jogador)
RAM	<i>Random Access Memory</i> (Memória de Acesso Aleatório)

SSD	<i>Solid-State Drive</i> (Unidade de Estado Sólido)
TADS	Tecnologia em Análise e Desenvolvimento de Sistemas
TCC	Trabalho de Conclusão de Curso
UCI	<i>Universal Chess Interface</i> (Interface Universal de Xadrez)
UI	<i>User Interface</i> (Interface do Usuário)
UML	<i>Unified Modeling Language</i> (Linguagem de Modelagem Unificada)
UX	<i>User Experience</i> (Experiência do Usuário)

SUMÁRIO

1	INTRODUÇÃO	15
1.1	Objetivo Geral	16
1.2	Objetivos específicos	16
1.3	Justificativa	16
1.4	Contribuições realizadas	17
1.5	Organização do trabalho	17
2	REVISÃO BIBLIOGRÁFICA	18
2.1	Fundamentação teórica	18
2.1.1	Xadrez Computacional e Protocolos (FEN e UCI)	18
2.1.2	Godot Engine e GDScript	19
2.1.3	Linguagem Python	19
2.1.4	Motor de Inteligência Artificial Stockfish	20
2.2	Trabalhos correlatos	21
2.2.1	Godot With Me: Chess (Repositório Open-Source)	21
2.2.2	Lichess (Plataforma Web Open-Source)	22
2.2.3	Lucas Chess	23
2.2.4	Chess.com	24
3	MATERIAIS E MÉTODOS	25
3.1	Materiais	25
3.1.1	Recursos de <i>Hardware</i>	25
3.1.2	Recursos de <i>Software</i>	26
3.2	Métodos	26
3.2.1	Levantamento de Requisitos e Modelagem	26
3.2.2	Etapas de Desenvolvimento	27
4	PROJETO DO SISTEMA	28
4.1	Estruturação do projeto	28
4.2	Componentes Visuais e Telas	29
4.2.1	Menu Principal e Opções	30
4.2.2	Tabuleiro de Jogo	30
4.3	Diagrama Entidade Relacionamento (DER)	31
4.4	Diagrama de Casos de Uso	32

4.5	Diagrama de Classes	33
4.6	Algoritmos Implementados	34
5	RESULTADOS E DISCUSSÃO	37
5.1	Apresentação dos componentes visuais	37
5.1.1	Menus e Customização	37
5.1.2	Tabuleiro e Mecânicas de Jogo	38
5.2	Avaliação de desempenho e estabilidade	39
5.3	Discussão dos resultados	39
6	CONSIDERAÇÕES FINAIS	41
6.1	Contribuições e Desafios de Integração	41
6.2	Trabalhos Futuros	42
	REFERÊNCIAS	43
	APÊNDICE A CÓDIGO-FONTE DO <i>MIDDLEWARE</i> PYTHON	45

1 INTRODUÇÃO

O xadrez é um dos jogos de tabuleiro mais antigos e estudados do mundo, sendo frequentemente utilizado ao longo da história como um laboratório para testar a capacidade lógica e o poder de processamento dos computadores. Desde a publicação do artigo seminal de Shannon (1950), o xadrez computacional tem servido como um campo fértil para explorar os limites da inteligência artificial (IA). Nesse sentido, segundo o Instituto de Estudos Avançados da USP (2025), o computador assume o papel de uma "máquina epistemológica", auxiliando na construção coletiva do conhecimento sobre o jogo e servindo de laboratório para algoritmos de alta complexidade.

Nos primórdios da ciência da computação, as partidas contra a máquina eram restritas a terminais de texto ou ambientes digitais extremamente rudimentares. No entanto, com a evolução tecnológica e a mudança nas expectativas dos usuários, os softwares de entretenimento passaram a exigir não apenas uma lógica matemática impecável, mas também uma camada de visualização imersiva e responsiva.

Atualmente, o desenvolvimento desse tipo de sistema envolve um desafio de engenharia de software bastante específico: integrar um arranjo visual atraente, rico em animações, a um núcleo de processamento capaz de calcular milhões de probabilidades matemáticas em poucos segundos.

Motores de jogos modernos, como a Godot Engine, são ferramentas excelentes para gerenciar sistemas interativos e efeitos sonoros. Contudo, não foram projetados para processar nativamente algoritmos densos de inteligência artificial. Por outro lado, ferramentas consagradas como o Stockfish são extremamente poderosas, mas operam exclusivamente via linha de comando, sem oferecer componentes de interface para o jogador humano.

Diante desse cenário, este trabalho apresenta o desenvolvimento do simulador digital *Royal Legacy*. O problema de pesquisa que norteia este projeto é: como integrar de forma eficiente um motor lógico de xadrez de alta complexidade (Stockfish) a uma estrutura gráfica moderna (Godot Engine), garantindo que o sistema seja visualmente personalizável e que os cálculos da máquina não comprometam a fluidez da experiência do usuário durante as partidas

1.1 Objetivo Geral

Este trabalho tem como objetivo desenvolver um simulador de xadrez digital interativo e customizável na Godot Engine, integrado ao motor de inteligência artificial Stockfish através de uma comunicação em linguagem Python, proporcionando uma experiência de jogo fluida, modular e com diferentes níveis de dificuldade.

1.2 Objetivos específicos

Para alcançar o objetivo geral, foram definidos os seguintes objetivos específicos:

- Implementar a camada gráfica e a lógica completa das regras do xadrez utilizando a linguagem nativa GDScript;
- Desenvolver um *middleware* em Python para estabelecer a comunicação assíncrona entre o *frontend* e o binário do Stockfish, operando através do protocolo *Universal Chess Interface* (UCI);
- Estruturar um sistema de gerenciamento de cenas e menus de opções, garantindo uma experiência de usuário (UX) intuitiva;
- Criar um sistema dinâmico de temas visuais baseado em dicionários de dados, permitindo a alteração estética em tempo real;
- Desenvolver um gerenciador de estados para controlar turnos e condições de fim de jogo, validando o fluxo correto das partidas nas modalidades Jogador contra Jogador (PvP) e Jogador contra Máquina (PvE).

1.3 Justificativa

A relevância deste trabalho reside na demonstração prática de uma arquitetura de software altamente modular. Conforme defende Fowler (2006), a separação entre a apresentação e o modelo é uma das heurísticas fundamentais do projeto de software, permitindo que a lógica de domínio evolua independentemente da interface. Este projeto propõe uma ponte técnica entre tecnologias de naturezas distintas, resolvendo uma lacuna comum: a dificuldade de executar processos pesados em segundo plano sem impactar a fluidez visual.

Ao documentar a separação dessas responsabilidades, este trabalho serve como um modelo de estudo valioso para a integração de motores visuais a inteligências artificiais complexas, demonstrando como mitigar o congelamento da interface gráfica por meio do processamento assíncrono (*multithreading*).

1.4 Contribuições realizadas

O desenvolvimento deste simulador trouxe contribuições práticas e metodológicas, destacando-se:

- **Arquitetura de Comunicação Assíncrona:** a implementação de um *script* em Python que atua como ponte provou ser viável para acoplar programas externos pesados à Godot sem causar interrupções (*stutters*) na interface principal;
- **Modularidade de Sistema:** a centralização da navegação por meio de um gerenciador global demonstrou a aplicação de boas práticas de *design*, facilitando a manutenção e evitando o acoplamento excessivo;
- **Escalabilidade Visual:** o uso de estruturas de dados flexíveis resultou em um código mais limpo e otimizado para futuras expansões de conteúdo.

1.5 Organização do trabalho

Este capítulo expôs o contexto inicial e os objetivos deste trabalho. O capítulo 2 apresenta os conceitos teóricos sobre o xadrez computacional e as ferramentas utilizadas. O capítulo 3 apresenta a metodologia e a arquitetura delineada. O capítulo 4 expõe o projeto de *software* construído, detalhando os *scripts* e o fluxo de dados. O capítulo 5 apresenta os resultados práticos obtidos. Por fim, o capítulo 6 compila as conclusões e sugere propostas para trabalhos futuros.

2 REVISÃO BIBLIOGRÁFICA

Este capítulo apresenta a base teórica e mercadológica que fundamenta o desenvolvimento do simulador *Royal Legacy*. Para compreender a estrutura do sistema, é necessário explorar os conceitos tecnológicos aplicados na sua construção, bem como analisar softwares já existentes que resolvem problemas semelhantes.

A seção 2.1 dedica-se à fundamentação teórica, descrevendo os conceitos de xadrez computacional, os protocolos de comunicação padronizados e as tecnologias (motores e linguagens de programação) escolhidas para o desenvolvimento. Em seguida, a seção 2.2 apresenta uma análise de trabalhos correlatos e plataformas de xadrez amplamente utilizadas no mercado, destacando as diferenças e justificando as escolhas de modelagem deste projeto em relação ao cenário atual.

2.1 Fundamentação teórica

A fundamentação teórica estabelece os conceitos e as ferramentas obtidos na literatura técnica que viabilizaram a construção do simulador. Segundo Wazlawick (2021), esta etapa é essencial para embasar cientificamente as decisões de projeto. A seguir, são detalhados os protocolos e *frameworks* centrais do trabalho.

2.1.1 Xadrez Computacional e Protocolos (FEN e UCI)

O desenvolvimento de ambientes visuais para xadrez exige a padronização da forma como o tabuleiro e as jogadas são representados pelo computador. A Notação Forsyth-Edwards (*FEN*, do inglês *Forsyth-Edwards Notation*) é o padrão global para descrever as posições de um jogo de xadrez em uma única linha de texto. Ela mapeia quais peças estão em quais casas, de quem é o turno e as disponibilidades de roque (Hooper; Whyld, 1992). Para alcançar alta eficiência no cálculo de movimentos, motores modernos utilizam *bitboards*. Segundo Pires e Pedrini (2022), tratam-se de estruturas de 64 bits que permitem operações lógicas rápidas diretamente no *hardware* do processador, representando o tabuleiro de forma otimizada.

Como exemplo prático, a configuração da posição inicial clássica de uma partida é traduzida pelo padrão FEN através da seguinte cadeia de caracteres:

`rnbqkbnr/pppppppp/8/8/8/8/PPPPPPPP/RNBQKBNR w KQkq - 0 1`

Nessa estrutura, o primeiro bloco mapeia as linhas do tabuleiro da oitava para a primeira fileira, separadas por barras; as letras maiúsculas denotam as peças brancas (ex: R para torre), as minúsculas indicam as pretas e os números representam espaços vazios consecutivos. Os blocos subsequentes definem, respectivamente: a cor ativa que deve jogar (w para brancas), os direitos de roque ainda vigentes (KQkq), a existência de alvos para capturas *en passant* (-), o contador de meio-lances para a regra de empate (0) e o número total de lances completos da partida (1).

Além da representação do tabuleiro, a comunicação com motores de Inteligência Artificial requer uma linguagem comum. O protocolo UCI (*Universal Chess Interface*) é um padrão aberto que permite que os sistemas gráficos (GUI) conversem com motores de xadrez de forma independente. O motor recebe a posição atual via texto e devolve a melhor jogada calculada, garantindo a separação absoluta entre a camada de apresentação e o núcleo de processamento lógico.

2.1.2 Godot Engine e GDScript

A Godot Engine é um motor de jogo (*Game Engine*) de código aberto e multiplataforma, amplamente reconhecido pela sua estrutura flexível baseada em nós (*nodes*) e cenas. Segundo Gregory (2018, p. 22, 38), um motor de jogo moderno define-se como "uma suite de ferramentas e tecnologias concebidas para simplificar o processo de desenvolvimento de jogos, fornecendo funcionalidades nucleares reutilizáveis". Essa robustez permite que o desenvolvedor foque na lógica específica, enquanto o motor gerencia a renderização complexa de GUI (Linietsky; Manzur; Godot Engine Community, 2023).

Além da base tecnológica, a qualidade da experiência do usuário foi central na concepção do *Royal Legacy*. Como afirma Norman (2013, p. 8), "a interface é o meio pelo qual o utilizador interage com o sistema e a sua clareza determina a eficácia dessa interação". Para a manipulação dessa lógica, utilizou-se o GDScript, a linguagem nativa da Godot, que facilita o acesso rápido aos elementos visuais e a manipulação de estruturas de dados (como os dicionários utilizados para a troca de temas).

2.1.3 Linguagem Python

Python é uma linguagem de programação de alto nível conhecida por sua legibilidade e vasta biblioteca padrão (Van Rossum; Drake, 2009). Neste projeto, atua como um *middleware*

(ponte) devido à sua robustez na manipulação de processos do sistema operacional. Sommerville (2011, p. 234) detalha a importância dessa estrutura em projetos de *software*:

O middleware é um software que fornece serviços de ligação a aplicações de software para além dos disponíveis no sistema operativo. Pode ser descrito como "cola de software", pois facilita a comunicação e a gestão de dados em sistemas distribuídos ou integrados.

Através de bibliotecas específicas, o *script* Python executa o binário da Inteligência Artificial Stockfish em segundo plano, recebe os comandos da camada gráfica da Godot, formata os dados e os repassa via protocolo UCI, resolvendo os problemas de travamento e garantindo a fluidez da renderização.

2.1.4 Motor de Inteligência Artificial Stockfish

O Stockfish é atualmente um dos motores de xadrez de código aberto mais fortes do mundo, operado inteiramente por linha de comando. O funcionamento lógico de motores desse porte baseia-se primordialmente no algoritmo **Minimax**, um modelo matemático de tomada de decisão para jogos de soma zero entre dois adversários (Max e Min). O objetivo fundamental do Minimax é mapear as jogadas futuras em uma estrutura de árvore, onde o jogador aliado tenta maximizar suas chances de vitória enquanto antecipa que o oponente escolherá a resposta que mais minimiza esses ganhos (Russell; Norvig, 2021).

No entanto, devido à explosão combinatória do xadrez, calcular todas as ramificações possíveis até o fim da partida torna-se computacionalmente inviável. Como explicam Russell e Norvig (2021, p. 165), no xadrez computacional, "a busca competitiva em jogos de soma zero exige algoritmos de poda para gerir a explosão combinatória de lances possíveis".

O sucesso do Stockfish deve-se justamente à utilização de uma versão altamente otimizada do algoritmo de busca alfa-beta (*Alpha-Beta Pruning*), uma técnica de otimização aplicada sobre a árvore de decisão do algoritmo Minimax. Esse mecanismo funciona estabelecendo dois valores de controle (alfa e beta) que representam as pontuações mínima e máxima que os jogadores podem garantir ao longo do cálculo. Caso o algoritmo detecte que uma jogada futura resultará em uma posição pior do que uma alternativa já avaliada, ele interrompe a exploração daquele caminho. Dessa forma, o sistema descarta ramos menos promissores da árvore de decisão sem perder a jogada ideal (Amorim, 2002).

Além disso, a partir da versão 12, o motor integra avaliações baseadas em Redes Neurais Eficientes Atualizáveis (*NNUE*, do inglês *Efficiently Updatable Neural Networks*). Ao contrário

das redes neurais profundas convencionais, que exigem placas gráficas de alto desempenho (GPUs) para processamento, a arquitetura NNUE é composta por conexões mais rasas e otimizadas para rodar em tempo real. O seu diferencial técnico reside na avaliação incremental: em vez de recalcular todo o tabuleiro a cada movimento, a rede atualiza apenas a matriz correspondente às peças que mudaram de posição. Essa abordagem híbrida permite ao motor avaliar milhões de posições por segundo com precisão sobre-humana, mesmo operando em unidades centrais de processamento (CPUs) comuns (Stockfish Team, 2023).

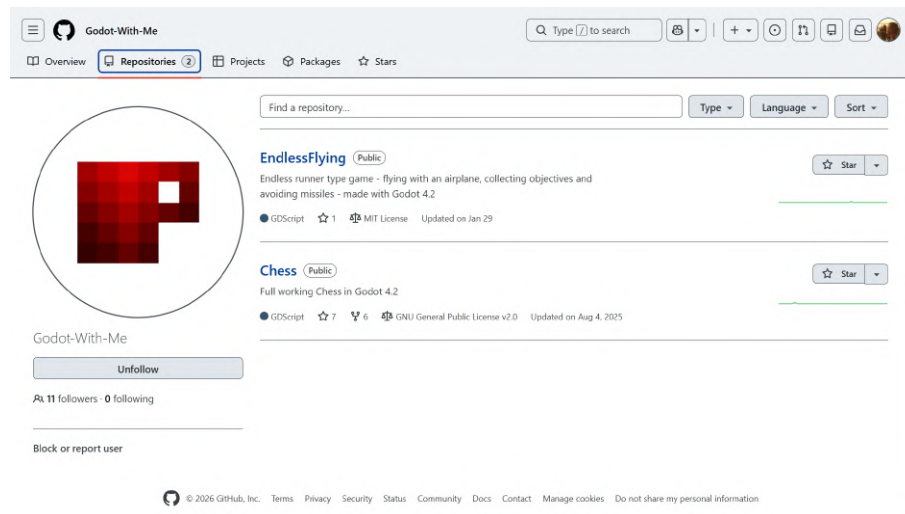
2.2 Trabalhos correlatos

Esta seção apresenta os principais *softwares*, plataformas e repositórios relacionados ao tema deste estudo. A análise destes sistemas permite não apenas identificar os padrões da indústria, mas também reconhecer as bases comunitárias que apoiaram a estruturação técnica do simulador *Royal Legacy*.

2.2.1 Godot With Me: Chess (Repositório Open-Source)

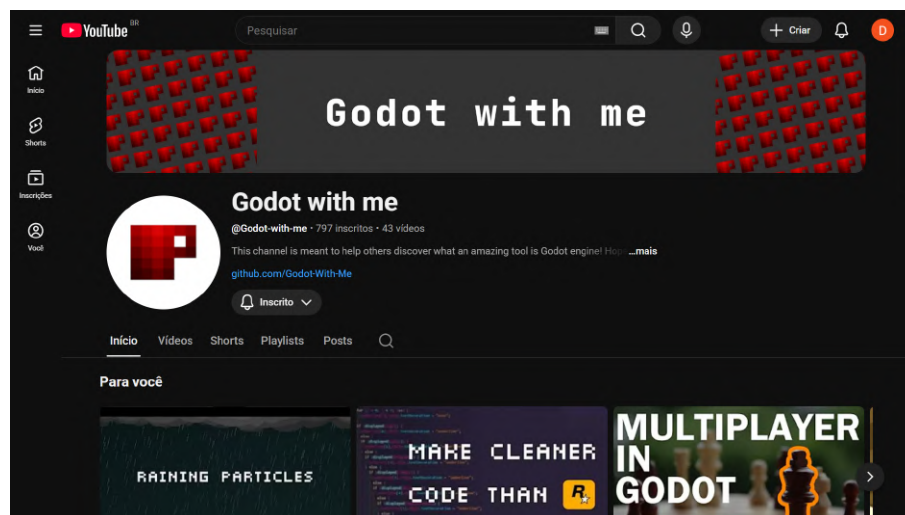
O projeto *Godot With Me: Chess* é um repositório hospedado no GitHub que serve como um guia prático para a implementação das regras básicas de xadrez utilizando a Godot Engine (Godot With Me, 2023a; Godot With Me, 2023b). A interface visual típica desse repositório de referência e a disposição inicial do seu canal no YouTube são apresentadas na Figura 1 e na Figura 2, respectivamente.

Este repositório serviu como referência inicial para o entendimento da manipulação de matrizes em GDScript. No entanto, o presente trabalho diferencia-se e expande significativamente o escopo sistêmico ao implementar a integração de uma IA externa operando em segundo plano via Python, a criação de um gerenciador de cenas para menus complexos e um sistema dinâmico de troca de temas visuais.

Figura 1 – Página do repositório *Open-Source*

Fonte: Repositório Godot With Me (2026)

Figura 2 – Visualização do canal no YouTube



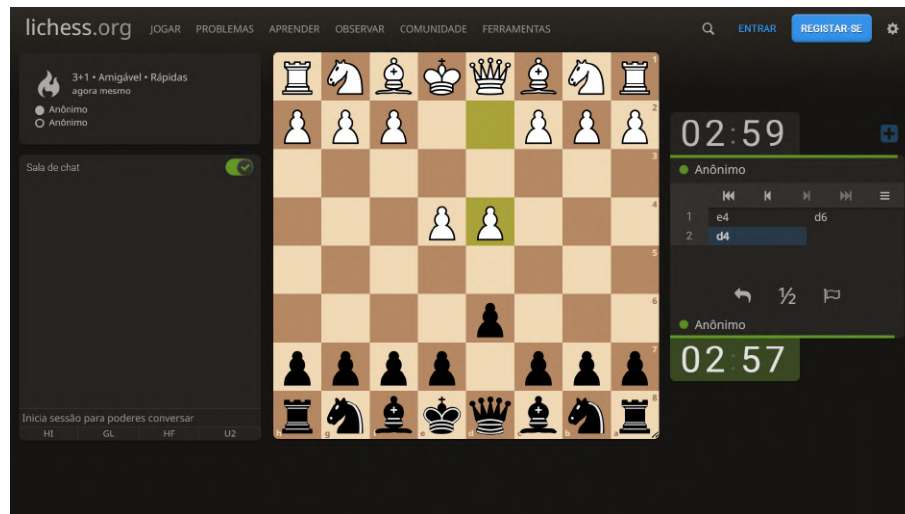
Fonte: Canal Godot With Me (2026)

2.2.2 Lichess (Plataforma Web Open-Source)

O Lichess é um dos maiores servidores de xadrez da Internet, sendo totalmente gratuito. A plataforma permite partidas utilizando o motor Stockfish nos seus servidores (*backend*) ou através do navegador do usuário (*frontend*).

A figura 3 apresenta o tabuleiro padrão da plataforma. Embora seja uma referência mundial, o Lichess possui um arranjo tecnológico focado em ambiente *web*. O presente trabalho difere por ser uma aplicação *desktop* nativa focada em renderização através de uma *Game Engine*, dispensando navegadores para a integração com a IA.

Figura 3 – Ambiente de partida contra o computador na plataforma Lichess



Fonte: Plataforma Lichess (2026)

2.2.3 Lucas Chess

O *Lucas Chess* é um *software desktop* voltado para o treinamento de xadrez, incluindo dezenas de motores de IA diferentes.

Assim como o projeto *Royal Legacy*, ele utiliza a integração local via protocolo UCI. No entanto, a sua camada de apresentação é construída com ferramentas tradicionais de janelas do sistema operacional, focando na utilidade estatística. O uso da Godot Engine neste trabalho de conclusão de curso (TCC) propõe justamente uma evolução audiovisual e interativa para esse tipo de simulador.

Figura 4 – Layout do software Lucas Chess

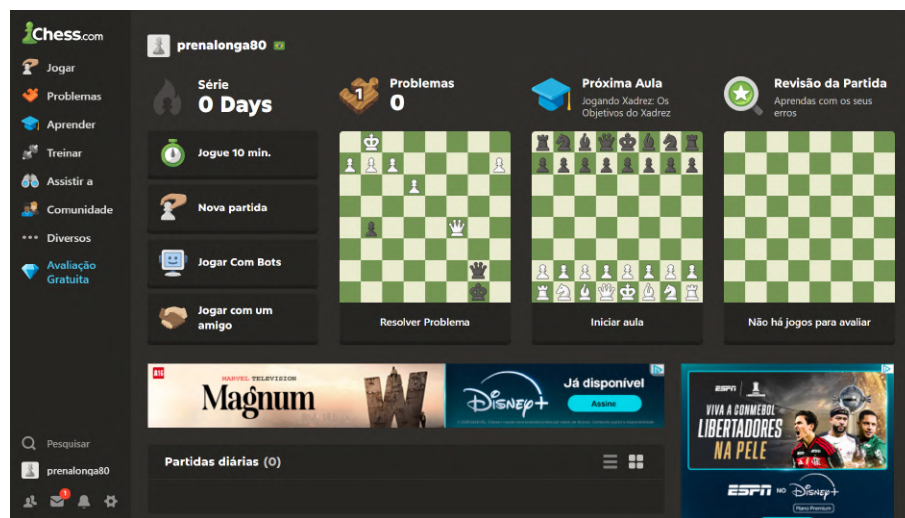


Fonte: Plataforma Lucas Chess (2026)

2.2.4 Chess.com

O Chess.com é a plataforma comercial líder de mercado. A aplicação oferece uma vasta gama de personalização de temas e avatares (*bots*). Por ser um *software* proprietário, os motores lógicos utilizados são fechados em servidores remotos. O desenvolvimento do *Royal Legacy* busca espelhar a experiência de customização estética e seleção de dificuldades encontrada no Chess.com, porém estruturando tudo de forma local, *offline* e utilizando apenas ferramentas *open-source*.

Figura 5 – Visual do site Chess.com



Fonte: Plataforma Chess.com (2026)

3 MATERIAIS E MÉTODOS

Este capítulo descreve a metodologia aplicada no desenvolvimento do simulador *Royal Legacy*. Serão apresentados os materiais tecnológicos, abrangendo *hardware* e *software*, e os métodos adotados para a modelagem, levantamento de requisitos e execução das etapas do projeto. Ressalta-se que a fundamentação teórica sobre o funcionamento dessas tecnologias foi abordada na seção 2.1.

3.1 Materiais

Os materiais empregados nesta pesquisa dividem-se em recursos físicos (*hardware*), utilizados para o desenvolvimento e aferição de desempenho, e lógicos (*software*), que compõem o ecossistema de programação e integração do sistema.

3.1.1 Recursos de *Hardware*

O desenvolvimento da aplicação, bem como os testes de estresse envolvendo o cálculo de jogadas pelo núcleo de Inteligência Artificial, foram conduzidos em uma estação de trabalho com as seguintes especificações:

- **Processador:** AMD Ryzen 5 4600G com Radeon Graphics (3.70 GHz), arquitetura baseada em x64, contando com 6 núcleos e 12 *threads*. Essa configuração foi essencial para suportar a execução do motor de jogo na via principal (*main thread*) em paralelo com o motor de xadrez em segundo plano, sem causar gargalos de processamento;
- **Memória RAM:** 16,0 GB de memória instalada (sendo 15,4 GB utilizáveis), garantindo a alocação necessária para a manipulação de matrizes do tabuleiro e o armazenamento da árvore de busca profunda do Stockfish;
- **Sistema Operacional e Arquitetura:** sistema operacional Windows de 64 bits, adequado para compilar o executável da Godot Engine e lidar com o alto volume de instruções da Inteligência Artificial;
- **Armazenamento:** unidade de armazenamento em disco de estado sólido (SSD), utilizada para reduzir o tempo de carregamento dos recursos visuais (*assets*) e acelerar as operações de leitura e escrita do *middleware* em Python.

3.1.2 Recursos de *Software*

As tecnologias e ferramentas de *software* foram aplicadas de forma modularizada, isolando a camada de apresentação da lógica computacional profunda:

- **Godot Engine e GDScript:** o motor gráfico atuou como o Ambiente de Desenvolvimento Integrado (IDE) principal. O GDScript foi a linguagem utilizada para programar as regras de movimentação, o gerenciamento de cenas e o controlador de áudio;
- **Python:** utilizado exclusivamente para a implementação do *middleware* (`chess_bridge.py`). A linguagem operou a biblioteca padrão `subprocess` para abrir fluxos de comunicação (leitura e escrita) com o motor de xadrez;
- **Stockfish:** empregado como o núcleo de processamento (*backend*) que recebe os estados do tabuleiro e devolve os cálculos matemáticos das melhores jogadas durante as partidas contra a máquina;
- **Visual Studio Code:** adotado como IDE auxiliar para a escrita, formatação e depuração do código Python, facilitando o monitoramento das saídas de terminal;
- **Protocolos FEN e UCI:** o padrão FEN serviu como estrutura de dados para trafegar o posicionamento das peças entre a Godot e o processo Python. O protocolo UCI funcionou como a interface de comunicação padrão para enviar comandos de configuração de dificuldade e receber as jogadas do Stockfish.

3.2 Métodos

O projeto foi conduzido sob a ótica da engenharia de *software*, utilizando a metodologia de desenvolvimento iterativo e incremental. Segundo Sommerville (2011), essa abordagem permite que o sistema seja construído em pequenos ciclos funcionais, garantindo que a base estrutural esteja consolidada antes da introdução de componentes de alta complexidade.

3.2.1 Levantamento de Requisitos e Modelagem

Dado que o projeto consiste na simulação de um jogo de tabuleiro clássico de regras estritas, o levantamento de requisitos não ocorreu por meio de entrevistas com usuários, mas sim através de pesquisa documental. Os requisitos lógicos de domínio foram extraídos do

manual oficial de regras da Federação Internacional de Xadrez (FIDE). Simultaneamente, os requisitos de integração basearam-se na documentação técnica do protocolo UCI.

Para a modelagem arquitetural do sistema, optou-se pela utilização de Diagramas de Fluxo de Dados (DFD) e fluxogramas lógicos. Esses artefatos mapearam o ciclo da informação desde a interação do usuário com a interface gráfica, passando pela tradução no *middleware* Python, até o processamento computacional no Stockfish e a consequente atualização do estado na tela.

3.2.2 Etapas de Desenvolvimento

A construção do simulador *Royal Legacy* foi fragmentada em quatro iterações principais, executadas de maneira sequencial:

1. **Iteração 1 - Estruturação Visual:** criação da grade do tabuleiro (*grid*), instanciamento das peças bidimensionais e implementação das lógicas basilares de seleção e movimentação *point-and-click*;
2. **Iteração 2 - Lógica de Domínio:** codificação das validações geométricas em GDScript (movimentos legais por tipo de peça) e implementação das regras especiais (*en passant*, roque e promoção) e condições de vitória (xeque e xeque-mate);
3. **Iteração 3 - Integração Assíncrona:** desenvolvimento da ponte em Python, codificação do mapeamento do tabuleiro para a notação FEN e consolidação da comunicação bidirecional com o Stockfish, assegurando a independência da *thread* principal da interface;
4. **Iteração 4 - Polimento e Componentização:** estruturação do gerenciador global de cenas (`estruturador.tscn`) para a navegação entre os menus, mapeado e disponível para consulta no repositório público do projeto (Royal Legacy, 2026). Esta etapa incluiu também a implementação do dicionário de dados para a transição dinâmica de temas visuais e calibração fina do sistema de áudio.

4 PROJETO DO SISTEMA

Este capítulo apresenta os principais elementos de projeto do simulador *Royal Legacy*. São detalhadas a estrutura do sistema, os protótipos das camadas visuais, a modelagem de dados para persistência, os diagramas estruturais e comportamentais baseados na *Unified Modeling Language* (UML), bem como a representação lógica dos algoritmos centrais de validação e comunicação.

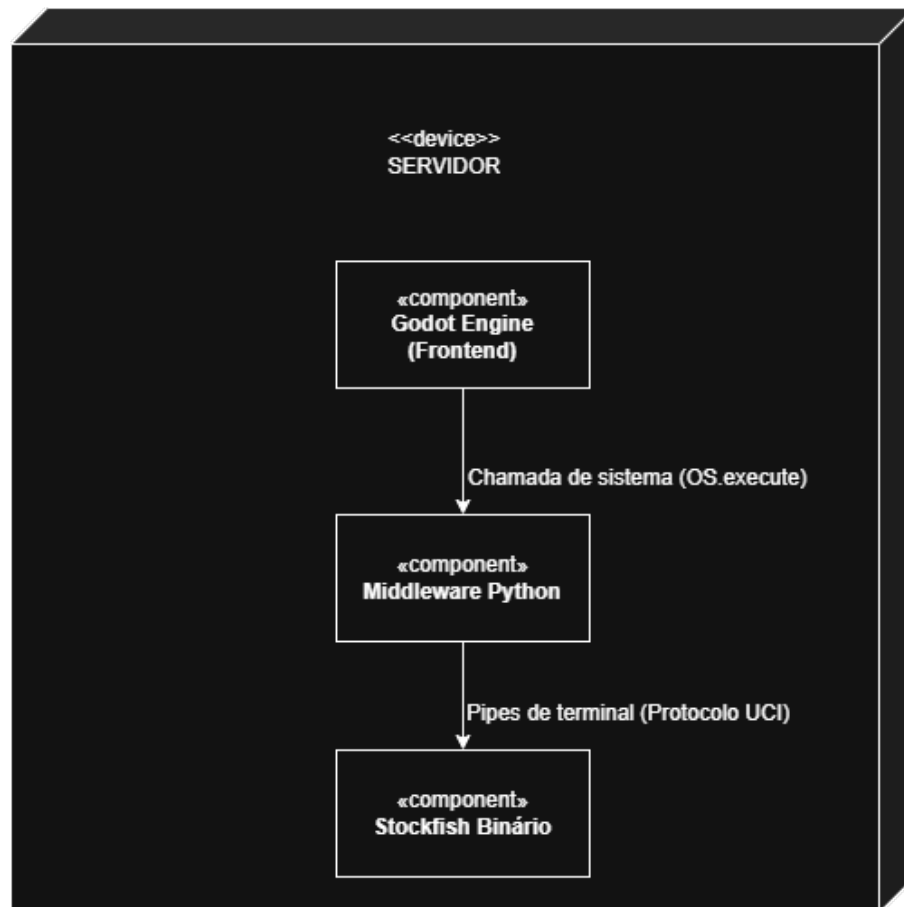
As decisões de projeto aqui expostas visam atender aos requisitos de modularidade, customização visual e execução assíncrona da Inteligência Artificial Stockfish, conforme justificado na seção 1.3.

4.1 Estruturação do projeto

O *Royal Legacy* foi projetado seguindo o princípio da separação de responsabilidades, dividindo o sistema em camadas distintas que se comunicam através de protocolos padronizados. A separação entre a camada de apresentação e a lógica de domínio é uma das heurísticas fundamentais do projeto de *software*. Segundo Fowler (2006), essa divisão permite que a lógica central (neste caso, o motor Stockfish) seja testada e evoluída independentemente do ambiente de interação visual.

A figura 6 apresenta o diagrama de implantação que ilustra como as partes do sistema foram divididas e como elas se comunicam.

Figura 6 – Diagrama de Implantação da estrutura técnica do Royal Legacy



Fonte: Elaborado pelo autor (2026)

O fluxo de comunicação segue o modelo Cliente-Servidor local:

1. **Camada de Apresentação (Godot Engine):** atua como o cliente (*Frontend*). Gerencia o componente visual, captura entradas do usuário e envia o estado do jogo (formato FEN) para o intermediário;
2. **Camada de Comunicação (Middleware Python):** um *script* que roda em segundo plano, agindo como ponte. Ele orquestra a execução da IA Stockfish e traduz os comandos;
3. **Camada Lógica (Stockfish Binary):** o núcleo de processamento (*Backend*). Recebe configurações via protocolo UCI e devolve os cálculos matemáticos profundos.

4.2 Componentes Visuais e Telas

Nesta seção são apresentadas as interfaces principais do *Royal Legacy*. Estas telas foram desenvolvidas com foco na disposição clara dos elementos e na Experiência do Usuário (UX).

4.2.1 Menu Principal e Opções

A tela do menu principal (figura 7) foi projetada para ser minimalista, oferecendo acesso rápido aos modos de jogo e configurações. A tela de opções (figura 8) apresenta os controles de volume (sons), a galeria de seleção de temas visuais e a seção de créditos do jogo.

Figura 7 – Layout do menu principal



Fonte: Royal Legacy (2026)

Figura 8 – Página de configurações de áudio, temas e créditos



Fonte: Royal Legacy (2026)

4.2.2 Tabuleiro de Jogo

O ambiente de jogo (figura 9) apresenta a grade 8×8 centralizada com peças em estilo *pixel art*, sobre um fundo verde-oliva. No canto superior esquerdo, encontram-se dois botões de controle de fluxo: um para retornar ao menu principal e outro com a função de desfazer jogadas ou reiniciar o tabuleiro.

Figura 9 – Ambiente final do tabuleiro com peças *pixel art* e botões de controle



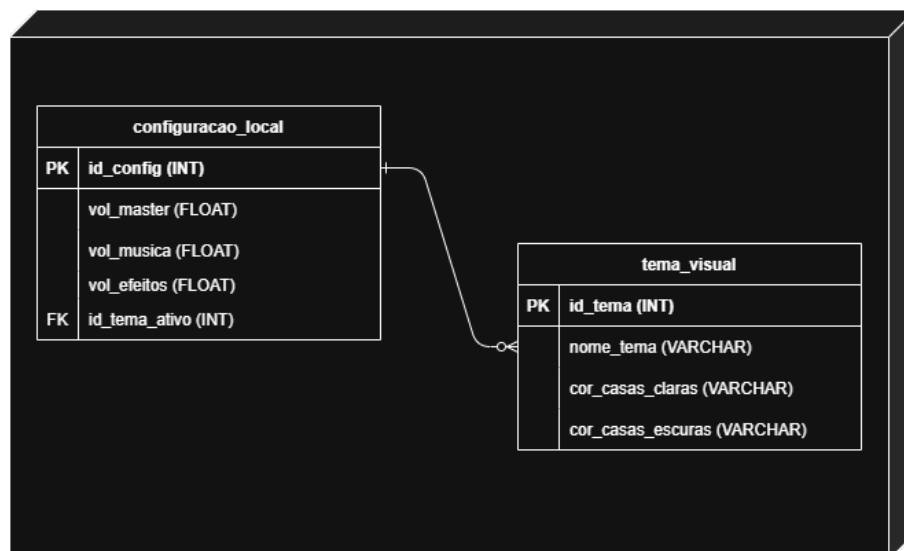
Fonte: Royal Legacy (2026)

4.3 Diagrama Entidade Relacionamento (DER)

Dado que o *Royal Legacy* é um simulador local focado na jogabilidade e não na gestão complexa de usuários, optou-se por não utilizar um banco de dados relacional tradicional, persistindo os dados em arquivos de configuração locais. Ainda assim, a modelagem segue as diretrizes da abordagem relacional. Como preconiza Date (2004), a organização em entidades permite manter a integridade lógica e a unicidade das informações persistidas.

A figura 10 apresenta o diagrama entidade relacionamento (DER) que rege a organização das preferências do sistema.

Figura 10 – Diagrama Entidade Relacionamento para persistência de preferências locais



Fonte: Elaborado pelo autor (2026)

O DER segue as normas formais de modelagem:

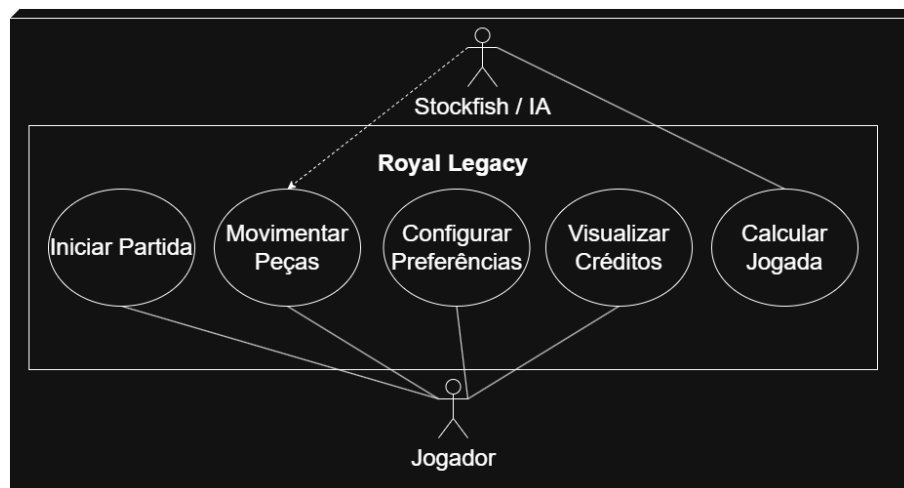
- **Configuracao_Local:** centraliza as preferências do usuário. Armazena de forma independente os níveis de volume para o canal mestre (*Master*), trilha sonora (*Musica*) e efeitos sonoros (*Efeitos*), além da referência ao tema visual ativo;
- **Tema_Visual:** estrutura os códigos de cores (casas claras e escuras) e identificadores de recursos visuais para a customização dinâmica do ambiente do tabuleiro.

Como o simulador prioriza sessões de jogo independentes, o modelo não contempla uma entidade de histórico de partidas, focando-se na persistência da experiência de uso e customização estética.

4.4 Diagrama de Casos de Uso

O diagrama de casos de uso (figura 11) descreve as funcionalidades oferecidas pelo simulador e as interações entre o usuário e o núcleo de IA.

Figura 11 – Diagrama de casos de uso do simulador



Fonte: Elaborado pelo autor (2026)

Os principais casos de uso identificados são:

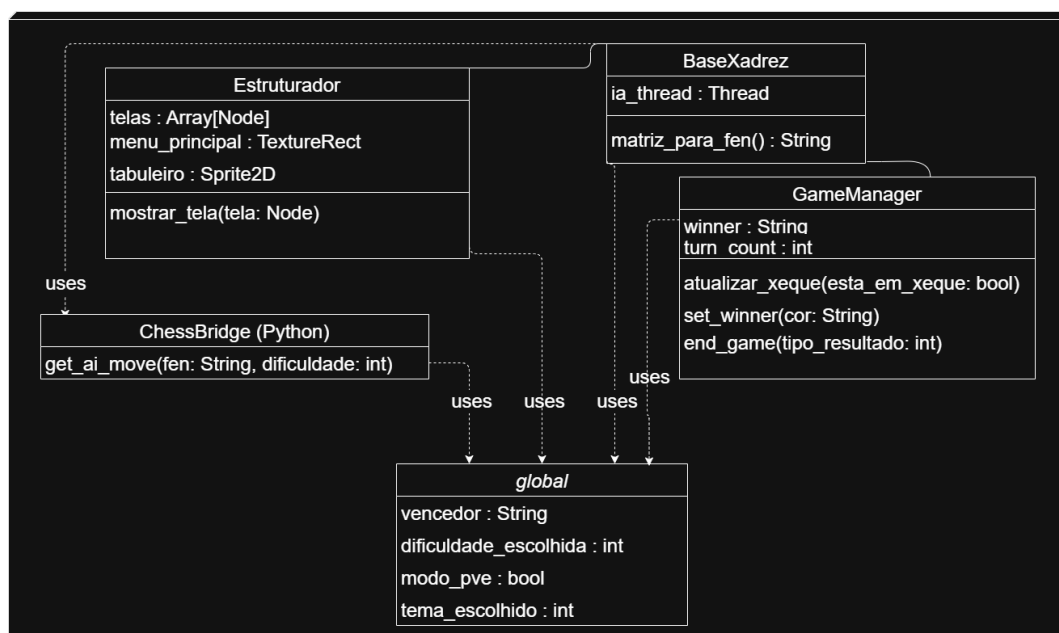
- **Configurar Preferências:** o usuário interage com as telas para ajustar volumes e a estética do jogo;
- **Iniciar Partida:** gerenciado pelo estruturador, permite definir o modo de jogo e o nível de dificuldade;

- **Movimentar Peças:** ação principal de interação no tabuleiro, cujas regras matemáticas e validações de lances (como movimentos válidos, capturas, xeque e roque) foram implementadas de forma nativa através de algoritmos customizados em *GDScript*, baseando-se no modelo lógico de matrizes bidimensionais estruturado no projeto pedagógico *Godot With Me* (Godot With Me, 2023b);
- **Calcular Jogada:** caso de uso executado pelo ator secundário (Stockfish) via *middleware*, disparado de forma assíncrona quando o modo *Player vs Environment* (PvE, ou seja, Jogador contra o Ambiente/Inteligência Artificial) está ativo, processando a resposta do motor gráfico e devolvendo a melhor jogada calculada.

4.5 Diagrama de Classes

A organização estrutural do código na Godot Engine e a integração com o *middleware* Python podem ser representadas formalmente. A modelagem estática do sistema, representada pelo diagrama de classes (figura 12), define a estrutura das classes, seus atributos e os relacionamentos de associação e composição necessários para o funcionamento do simulador (Massollar, 2005).

Figura 12 – Diagrama de classes detalhando a estrutura em GDScript e Python



Fonte: Elaborado pelo autor (2026)

Destacam-se as seguintes classes no funcionamento do sistema:

- **Global:** script configurado por meio do recurso *autoload* da Godot Engine, responsável por persistir o estado da aplicação e reter os dados globais de configuração do simulador durante a alternância de cenas;
- **Estruturador:** atua como o nó raiz, instanciando e gerenciando a transição entre os componentes gráficos através do método `mostrar_tela()`;
- **BaseXadrez:** classe central do jogo. Detém a matriz do tabuleiro, executa a validação de regras e possui o método `matriz_para_fen()`, essencial para traduzir o estado do jogo. Também gerencia a *thread* da IA;
- **GameManager:** classe responsável por monitorar o fluxo da partida, emitindo sinais de xeque e definindo o encerramento do jogo;
- **ChessBridge:** *script* Python independente que recebe a *string* FEN e a dificuldade, instanciando o motor Stockfish para calcular e retornar a jogada.

4.6 Algoritmos Implementados

Nesta seção, detalha-se a lógica de conversão do estado do jogo e a comunicação com o motor de análise. O algoritmo 1 apresenta a tradução da matriz interna da Godot para a notação FEN, essencial para que o Stockfish compreenda a posição das peças.

Algoritmo 1: Lógica de Conversão da Matriz do Tabuleiro para Notação FEN.

```

Entrada: matriz_do_tabuleiro

1  início
2      texto_fen ← texto vazio;
3      para cada linha da matriz_do_tabuleiro faça
4          casas_vazias ← 0;
5          para cada coluna da linha atual faça
6              peca_atual ← verificar a peça na posição atual;
7              se peca_atual == null então
8                  casas_vazias ← casas_vazias + 1;
9              senão
10                 se casas_vazias > 0 então
11                     concatenar o número de casas_vazias ao texto_fen;
12                     casas_vazias ← 0;
13                 fim
14                 letra_pecas ← converter a peca_atual para sua letra correspondente;
15                 concatenar letra_pecas ao texto_fen;
16             fim
17         fim
18         se casas_vazias > 0 então
19             concatenar o número de casas_vazias ao texto_fen;
20         fim
21         se linha < 8 então
22             concatenar o símbolo "/" ao texto_fen;
23         fim
24     fim
25     retorna texto_fen;
26 fim

```

Fonte: Elaborado pelo autor (2026)

O processo de decisão da IA Stockfish é executado em uma *Thread* separada para evitar o congelamento da camada visual. O algoritmo 2 descreve a lógica do *middleware*. É importante notar que a aplicação do limite de profundidade (linhas 10 e 13) é uma execução prática

do modelo *Minimax* estruturado, conforme descrito por Pires e Pedrini (2022), garantindo a limitação da árvore de pesquisa e o tempo de resposta aceitável da máquina.

Algoritmo 2: Lógica de Processamento de Movimento da IA Stockfish.

Entrada: estado_do_tabuleiro, nivel_dificuldade

```

1  início
2      ler o estado_do_tabuleiro;
3      se nivel_dificuldade = 1 então
4          lista_movimentos ← obter todos os movimentos válidos;
5          movimento_escolhido ← sortear aleatoriamente um movimento da
              lista_movimentos;
6      senão
7          iniciar o motor de xadrez;
8          se nivel_dificuldade = 2 então
9              limite_busca ← profundidade 1;
10             senão
11                 limite_busca ← profundidade 20;
12             fim
13             movimento_escolhido ← calcular a melhor jogada usando o limite_busca;
14      fim
15      resposta ← formatar movimento_escolhido e status de sucesso;
16      retorna resposta;
17 fim
  
```

Fonte: Elaborado pelo autor (2026)

5 RESULTADOS E DISCUSSÃO

Este capítulo apresenta e debate os resultados obtidos após a implementação do simulador *Royal Legacy*. São expostas as camadas visuais finais, traçando um comparativo com os protótipos idealizados, além da apresentação das métricas de desempenho que validam a eficiência da comunicação assíncrona e do sistema de temas dinâmicos.

5.1 Apresentação dos componentes visuais

A camada de apresentação foi refinada na Godot Engine com a aplicação de uma estética *pixel art* coerente e paletas de cores harmônicas. O resultado final priorizou a clareza visual, requisito essencial para um jogo de tabuleiro.

5.1.1 Menus e Customização

As figuras 13 e 14 apresentam os ambientes dedicados à customização da experiência do usuário.

Utilizando o *script* `tela_temas.gd` (Royal Legacy, 2026) (figura 13) para alternar instantaneamente entre os recursos visuais (*assets*), como o tema "Clássico" e "Moderno". A modularização desta tela garante que a escolha seja salva globalmente e aplicada ao tabuleiro de forma dinâmica, sem necessidade de recarregar o sistema.

Figura 13 – Ambiente final de seleção de temas visuais



Fonte: Royal Legacy (2026)

Por sua vez, o menu de configurações de áudio (figura 14), gerenciado pelo *script* `tela_sons.gd` (Royal Legacy, 2026), permite o controle granular dos decibéis. A separação dos canais em *Master*, Música e Efeitos proporciona ao jogador uma experiência auditiva totalmente personalizada, sem interferir no fluxo do jogo.

Figura 14 – Painel final de controle de canais de áudio

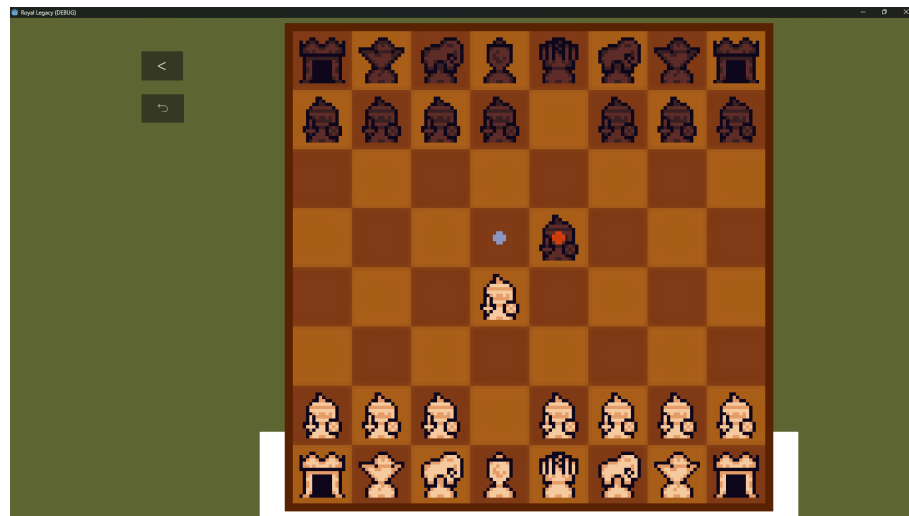


Fonte: Royal Legacy (2026)

5.1.2 Tabuleiro e Mecânicas de Jogo

A figura 15 exibe a tela de jogo principal. O refinamento gráfico incluiu a renderização de peças personalizadas e a implementação de sinais visuais de seleção. O uso do *script* `base_xadrez.gd` (Royal Legacy, 2026) permitiu que a lógica de movimentação respondesse de forma fluida, com o realce das casas de origem e destino, facilitando a compreensão do fluxo da partida pelo usuário.

Figura 15 – Ambiente de partida com aplicação de temas e realce de movimentos



Fonte: Royal Legacy (2026)

5.2 Avaliação de desempenho e estabilidade

Para validar a solução de integração assíncrona, o sistema foi testado monitorando-se a taxa de quadros (*Frames Per Second* - FPS) da Godot enquanto o *middleware* Python processava as jogadas da IA. Segundo Sommerville (2011), o desempenho contínuo e a confiabilidade de resposta são requisitos não-funcionais críticos para a aceitação de sistemas interativos complexos.

Os testes foram divididos nos três níveis de dificuldade programados em `chess_bridge.py`. Os resultados, consolidados na tabela 1, demonstram o tempo médio entre o envio da *string* FEN e o recebimento da jogada UCI, garantindo uma taxa de atualização de tela impecável.

Tabela 1 – Métricas de tempo de resposta da IA Stockfish e estabilidade da renderização

Dificuldade	Lógica da IA Stockfish	Tempo de Resposta	Estabilidade (FPS)
Fácil	Movimento Aleatório	0,50 segundos*	Constante a 60 FPS
Médio	Stockfish (Depth 1)	0,85 segundos	Constante a 60 FPS
Difícil	Stockfish (Depth 20)	2,10 segundos	Constante a 60 FPS

*Inclui o atraso proposital de *sleep* para simulação de pensamento.

Fonte: Elaborado pelo autor (2026).

5.3 Discussão dos resultados

A análise dos dados confirma que o principal problema técnico delimitado no início da pesquisa o congelamento da tela durante o processamento de cálculos profundos foi mitigado

com sucesso. O uso do comando `ia_thread.start()` no *script* principal (Royal Legacy, 2026) permitiu que, mesmo quando o motor Stockfish operava em profundidade máxima (nível Difícil), o motor gráfico da Godot continuasse processando as entradas do usuário e as animações sem interrupções na *thread* principal.

No nível fácil, o tempo de resposta de 0,5 segundos foi mantido por meio de uma pausa programada (*time.sleep*) no Python (apêndice: A). Essa decisão técnica visa humanizar a experiência do usuário. Como argumenta Norman (2013, p. 8), o *design* de interação deve alinhar-se à expectativa humana; uma resposta artificial imediata em jogos de tabuleiro pode causar desconforto e quebra de imersão para jogadores iniciantes.

Nos níveis superiores, o *middleware* provou ser resiliente ao gerenciar a comunicação via *pipes* (dutos) de terminal, sem causar vazamento de memória ou picos de processamento na linha de execução principal (*main thread*). Para viabilizar essa arquitetura entre o ecossistema do motor gráfico e o código de inteligência artificial, utilizou-se o conceito *Inter-Process Communication*, cujo algoritmo completo desenvolvido encontra-se disponível no Apêndice A. Os *pipes* operam como canais de fluxo de dados gerenciados pelo sistema operacional, permitindo que a saída de texto da interface do jogo seja diretamente conectada à entrada do motor de análise Stockfish, garantindo uma troca de dados contínua e de baixa latência.

Por fim, a modularização do projeto em cenas independentes mostrou-se eficiente para a otimização de recursos, mantendo o consumo da memória de Acesso Aleatório (RAM) estável, uma vez que elementos densos, como o tabuleiro, só são instanciados após a seleção do modo de jogo. Esses resultados validam as escolhas de estruturação do sistema baseadas nos princípios de separação de responsabilidades (Fowler, 2006), conforme o proposto no capítulo 4.

6 CONSIDERAÇÕES FINAIS

Este capítulo apresenta as considerações finais do trabalho, resumizando os resultados obtidos com o desenvolvimento do simulador *Royal Legacy*. O objetivo geral desta pesquisa foi plenamente alcançado, uma vez que foi possível validar a viabilidade técnica de integrar um motor de análise complexo (Stockfish) a um motor de jogos moderno (Godot), garantindo tanto a precisão lógica quanto a fluidez visual. O projeto resultou em uma experiência de xadrez digital estável, integrando o processamento de alto desempenho sem comprometer a camada de visualização. O capítulo divide-se nas seções 6.1, que detalha os avanços técnicos e desafios superados, e 6.2, que sugere caminhos para a continuidade desta pesquisa.

6.1 Contribuições e Desafios de Integração

O desenvolvimento deste projeto resultou em avanços técnicos e estruturais significativos para a criação de sistemas de jogos integrados a motores externos. As principais contribuições e superações técnicas deste trabalho são detalhadas a seguir:

- **Arranjo Assíncrono de Comunicação:** a criação de um *middleware* em Python demonstrou ser uma solução eficaz para mitigar a latência na camada gráfica (comumente referida como *stuttering*). A execução isolada do binário do Stockfish permitiu que o ambiente de interação mantivesse uma taxa de renderização constante de 60 FPS, mesmo durante o processamento de jogadas complexas;
- **Motor Lógico Independente em GDScript:** a transcrição integral das regras do xadrez incluindo movimentos especiais como *en passant*, roque e promoção para a linguagem nativa da Godot resultou em um sistema modular. O uso de matrizes bidimensionais para a representação do tabuleiro facilitou a manutenção do código e a validação contínua dos estados de jogo;
- **Superação de Desafios na Integração de Sistemas:** um dos principais obstáculos superados foi a sincronização entre o interpretador Python e o sistema de sinais da Godot. Foi necessária uma gestão rigorosa de processos para impedir que instâncias do motor de busca permanecessem executando em segundo plano como processos órfãos, garantindo a estabilidade e a integridade da memória do sistema durante longas sessões de uso;

- **Sistema de Customização Dinâmica:** a implementação de um gerenciador de temas baseado em dicionários de dados otimizou a transição de recursos visuais (*sprites* e paletas) em tempo de execução. Esta abordagem eliminou a necessidade de estruturas condicionais extensas, tornando o componente visual mais leve e eficiente em termos de consumo de memória.

6.2 Trabalhos Futuros

Apesar do pleno funcionamento da estrutura sistêmica proposta e do atingimento dos objetivos iniciais, o escopo do *Royal Legacy* permite diversas expansões. Para que futuros pesquisadores ou desenvolvedores continuem contribuindo com o tema, sugerem-se as seguintes melhorias a médio e longo prazo:

- **Implementação de *Multiplayer Online*:** expandir o sistema para suportar partidas remotas entre jogadores (PvP via rede), utilizando a API de rede de alto nível da Godot ou *WebSockets* para a troca eficiente de *strings* FEN através de um servidor central;
- **Portabilidade para Dispositivos Móveis:** adaptar a camada de apresentação para telas sensíveis ao toque (UX/UI *mobile*) e reestruturar a comunicação assíncrona para compilar o código-fonte do Stockfish nativamente como um módulo em C++ (GDExtension), permitindo a execução em sistemas Android e iOS sem a necessidade do interpretador Python;
- **Integração com Múltiplos Motores (IA):** adicionar suporte a outras Inteligências Artificiais de xadrez de código aberto compatíveis com o protocolo UCI, como a Leela Chess Zero (baseada em redes neurais), permitindo que o usuário enfrente oponentes virtuais com diferentes "personalidades" e estilos de jogo;
- **Análise de Partidas em Tempo Real:** desenvolver um módulo gráfico que traduza a avaliação numérica contínua do Stockfish em uma barra de vantagens visual durante o jogo, auxiliando o usuário no aprendizado, no estudo de posições e na detecção de erros graves (*blunders*).

REFERÊNCIAS

- Amorim, C. A. de. **A Máquina e seus Limites: Uma Investigação Sobre o Xadrez Computacional**. Dissertação (Mestrado) — UFBA, 2002. Citado na página 19.
- Date, C. J. **Introdução a Sistemas de Bancos de Dados**. 8. ed. Rio de Janeiro: Campus, 2004. Citado na página 30.
- Fowler, M. **Padrões de Arquitetura de Aplicações Corporativas**. Porto Alegre: Bookman, 2006. Citado na página 15, 27 e 39.
- Godot With Me. **Desenvolvimento de Jogos com Godot Engine**. 2023. Citado na página 20. Disponível em: <https://github.com/Godot-With-Me/Chess>. Acesso em: 31 mar. 2026.
- Godot With Me. **Tutorial de Criação de Xadrez na Godot**. 2023. Citado na página 20. Disponível em: <https://youtu.be/1y57hJo1ONQ?si=IkjlcUxRxrgqTy1Q>. Acesso em: 31 mar. 2026.
- Gregory, J. **Game Engine Architecture**. 3. ed. Boca Raton: CRC Press, 2018. Citado na página 18.
- Hooper, D.; Whyld, K. **The Oxford Companion to Chess**. 2. ed. Oxford: Oxford University Press, 1992. Citado na página 17.
- Instituto de Estudos Avançados da USP. O computador como máquina epistemológica: o caso do xadrez computacional. **IEA-USP**, 2025. Citado na página 14.
- Linietzky, J.; Manzur, A.; Godot Engine Community. **Godot Engine Documentation**. 2023. Citado na página 18. Disponível em: <https://docs.godotengine.org/>. Acesso em: 15 mar. 2026.
- Massollar, J. **Modelagem de Sistemas com UML**. Rio de Janeiro: COPPE/UFRJ, 2005. Citado na página 32.
- Norman, D. A. **The Design of Everyday Things**. New York: Basic Books, 2013. Citado na página 18.
- Pires, G. S.; Pedrini, H. Métodos de aprendizado de máquina aplicados ao jogo de xadrez. **IC/UNICAMP**, 2022. Citado na página 17.
- Royal Legacy. **Desenvolvimento de um Simulador de Xadrez Com Integração do Motor Stockfish Utilizando Godot**. 2026. Citado na página 26, 36, 37 e 39. Disponível em: https://github.com/Joao-Paulo06/Royal_Legacy. Acesso em: 15 mar. 2026.

Russell, S.; Norvig, P. **Artificial Intelligence: A Modern Approach**. 4. ed. Hoboken: Pearson, 2021. Citado na página 19.

Sommerville, I. **Engenharia de Software**. 9. ed. São Paulo: Addison-Wesley, 2011. Citado na página 19, 25 e 38.

Stockfish Team. **Stockfish: Open Source Chess Engine**. 2023. Citado na página 20. Disponível em: <https://stockfishchess.org/>. Acesso em: 15 mar. 2026.

Van Rossum, G.; Drake, F. L. **Python 3 Reference Manual**. Scotts Valley: CreateSpace, 2009. Citado na página 18.

Wazlawick, R. S. **Metodologia de Pesquisa em Ciência da Computação**. 3. ed. Rio de Janeiro: LTC, 2021. Citado na página 17.

APÊNDICE A – CÓDIGO-FONTE DO *MIDDLEWARE* PYTHON

Código A.1 – Código-fonte do script de integração em Python (middleware).

```
1  import sys
2  import json
3  import chess
4  import chess.engine
5  import os
6  import random
7  import time
8
9  def get_ai_move(fen, dificuldade_str):
10     diretorio_atual = os.path.dirname(os.path.abspath(__file__))
11     caminho_stockfish = os.path.join(diretorio_atual, "stockfish
        .exe")
12     board = chess.Board(fen)
13
14     try:
15         dificuldade = int(dificuldade_str)
16
17         if dificuldade == 1:
18             # Pega todas as jogadas validas no tabuleiro agora
19             movimentos_legais = list(board.legal_moves)
20
21             # Escolhe uma jogada de forma aleatoria (Modo Caos)
22             movimento_escolhido = random.choice(movimentos_legais)
23
24             time.sleep(0.5)
25
26         print(json.dumps({
27             "status": "success",
28             "move_uci": movimento_escolhido.uci(),
```

```

29         "debug_info": "IA Nivel 1"
30     )))
31     return
32
33     # ===== NIVEIS MEDIO E DIFICIL =====
34     with chess.engine.SimpleEngine.popen_uci(caminho_stockfish)
35         as engine:
36
37         if dificuldade == 2:
38             # MEDIO: Jogador de clube amador.
39             engine.configure({"Skill Level": 1})
40             limite = chess.engine.Limit(depth=1, time=0.5)
41
42         else:
43             # DIFICIL (3): Grande Mestre implacavel.
44             engine.configure({"Skill Level": 20})
45             limite = chess.engine.Limit(depth=20, time=1.5)
46
47         result = engine.play(board, limite)
48
49         print(json.dumps({
50             "status": "success",
51             "move_uci": result.move.uci(),
52             "debug_info": f"IA Nivel {dificuldade}"
53         })))
54
55     except Exception as e:
56         print(json.dumps({
57             "status": "error",
58             "message": str(e)
59         })))

```

```
60 # Logica para receber os comandos da Godot
61 if __name__ == "__main__":
62     if len(sys.argv) > 3:
63         comando = sys.argv[2]
64         if comando == "get_ai_move":
65             fen_board = sys.argv[1]
66             nivel_dificuldade = sys.argv[3]
67             get_ai_move(fen_board, nivel_dificuldade)
```