DESENVOLVIMENTO DE JOGO DE CARTAS COM BOT EM PYTHON NO CONTEXTO DO PROJETO LACIA

Fernanda Oliveira Cardoso[1]

Rodrigo Grassi Martins[2]

[1]Estudante de Ciência da Computação, IFTM, Campus Ituiutaba, MG,

fernanda.oc@estudante.iftm.edu.br

[2]Professor do IFTM, Campus Ituiutaba, rodrigograssi@iftm.edu.br

Resumo: Este trabalho teve como objetivo o desenvolvimento e a implementação de um Jogo da Memória digital em Python, inserido no contexto do Projeto LACIA. A problemática central envolveu a criação de um bot autônomo com Inteligência Artificial (IA), baseado em uma memória simulada de curto prazo, que oferecesse uma interação desafiadora contra o jogador. A metodologia utilizou a biblioteca Pygame para a interface gráfica e implementou um algoritmo de decisão que permite ao bot memorizar cartas reveladas para otimizar suas jogadas. Como resultado, foi entregue um software de jogo funcional e completo, que inclui três temas (Super Mario, Pokémon e Zelda) e telas de resultado personalizadas. Os testes de jogabilidade validaram que o bot se comporta como um oponente coerente, capaz de competir e até vencer o jogador humano, consolidando-se como uma plataforma robusta para futuros estudos de IA aplicada a jogos no âmbito do LACIA.
Palavras-chave: bot; jogos de cartas; python; inteligência artificial; LACIA.

Abstract: This work aimed at the development and implementation of a digital Memory Game in Python, inserted in the context of the LACIA Project. The central problem involved the creation of an autonomous bot with Artificial Intelligence (AI), based on a simulated short-term memory, designed to offer a challenging interaction against the player. The methodology used the Pygame library for the graphical interface and implemented a decision algorithm that allows the bot to memorize revealed cards to optimize its moves. As a result, a functional and complete game software was delivered, which includes three themes (Super Mario, Pokémon, and Zelda) and personalized result screens. Playability tests validated that the bot behaves as a coherent opponent, capable of competing and even defeating the human player, establishing itself as a robust platform for future studies of applied AI within the scope of LACIA.
Keywords: bot; card games; python; artificial intelligence; LACIA.

1 INTRODUÇÃO

As tecnologias interativas, com destaque para os jogos digitais, consolidaram-se como ferramentas de grande relevância no panorama educacional contemporâneo (ALVES; COUTINHO, 2020). Este fenômeno se intensificou à medida que as instituições reconhecem o potencial lúdico e engajador dos jogos como catalisadores do aprendizado. Tais recursos se alinham diretamente às metodologias ativas de aprendizagem, filosofia que busca posicionar o estudante no centro do processo educativo, estimulando a participação ativa, o pensamento crítico e a resolução de problemas (BARBOSA; MOURA, 2013).

Nesse sentido, a combinação da criação de jogos com linguagens de programação acessíveis, como Python, revela-se uma estratégia promissora. A escolha por Python deve-se à sua sintaxe clara e legível, o que a torna ideal para o ensino introdutório de lógica de programação e desenvolvimento de software (BORGES, 2014; REBOUÇAS et al., 2013). É nesse cenário que os jogos de cartas, com suas regras estruturadas e demanda por pensamento estratégico, apresentam-se como um campo fértil para a aplicação de conceitos de lógica e Inteligência Artificial (IA) (MUNÁRRIZ, 1994).

Este projeto, em particular, insere-se no contexto do Laboratório de Computação Integrada e Aplicada (LACIA), um projeto de extensão universitária que visa promover o ensino de programação e o desenvolvimento de software. A problemática central que motivou este estudo foi o desafio de criar um agente computacional (bot) para um Jogo da Memória que ultrapassasse a simples aleatoriedade, buscando oferecer uma interação desafiadora e "humana", demandando a aplicação de técnicas de IA específicas para jogos (FILHO; ANDRADE; SOUZA, 2022).

1.1 Justificativa Técnica e Contribuição

A escolha da linguagem Python e da biblioteca Pygame justifica-se pela robustez e pela facilidade em prototipar interfaces gráficas e lógicas de decisão. Python, sendo uma linguagem de alto nível, permite a escrita de códigos mais concisos, o que acelera o ciclo de desenvolvimento em projetos de pequena e média escala. O Pygame, em particular, é uma biblioteca multiplataforma e de código aberto, amplamente documentada e recomendada para o desenvolvimento de jogos 2D (PYGAME, 2024). A metodologia de implementação seguiu padrões práticos de desenvolvimento com esta biblioteca, como a gestão de um loop de jogo principal e o tratamento eficiente de eventos do usuário (SWEIGART, 2012).

A inovação central deste trabalho reside no desenvolvimento de um bot capaz de tomar decisões coerentes, simulando o processo de memorização e esquecimento. Este sistema de memória simulada de curto prazo, com capacidade limitada, foi intencionalmente projetado para evitar que o oponente fosse infalível, elevando o nível de desafio de forma realista. Este modelo representa a principal contribuição técnica desta pesquisa e estabelece uma base robusta para futuros estudos em IA aplicada a jogos no âmbito do LACIA.

1.2 Objetivos

O planejamento deste trabalho foi guiado por um objetivo central e desdobrado em metas específicas, visando a entrega de um produto funcional e tecnicamente inovador. O objetivo principal deste trabalho foi desenvolver e implementar um Jogo da Memória digital completo, dotado de um agente computacional autônomo. Para alcançar tal finalidade, foram estabelecidos os seguintes objetivos específicos:

- Fundamentar-se teoricamente no desenvolvimento de jogos utilizando a linguagem Python;
- Modelar a lógica de turnos e as regras clássicas do Jogo da Memória;
- Implementar uma Inteligência Artificial com memória de curto prazo limitada para simular um comportamento realista e desafiador;
- Criar uma interface gráfica funcional com o suporte a múltiplos temas (Super Mario, Pokémon e Zelda), garantindo a atratividade visual;
- Validar o desempenho do bot em partidas reais contra jogadores humanos, comprovando a coerência e o equilíbrio do nível de desafio proposto.

2 DESENVOLVIMENTO

O desenvolvimento do software foi pautado pela metodologia de prototipação rápida, utilizando as ferramentas Python e Pygame, e buscando conciliar as necessidades pedagógicas com as melhores práticas de engenharia de software. A arquitetura foi definida com foco na modularidade e manutenibilidade do código, permitindo que a lógica, a interface e as regras do jogo fossem desenvolvidas e testadas de forma independente.

2.1 ARQUITETURA DO SISTEMA E METODOLOGIA

2.1.1 Arquitetura Modular do Software

A estrutura do software é composta por três núcleos principais que garantem a segregação das responsabilidades e a manutenibilidade do projeto. Esta arquitetura orientada a módulos facilitou a manutenção e a rastreabilidade de bugs.

1. **Núcleo de Inteligência (IA):** É o componente de *backend* responsável por toda a lógica de tomada de decisão do agente computacional. É encapsulado no módulo `bot_memoria.py`, que contém a classe `BotMemoria` e o algoritmo de memória de curto prazo.

2. **Núcleo de Interface e Fluxo (GUI):** Gerencia as telas da aplicação através de um conjunto de módulos sequenciais: `menu.py` (ponto de entrada), `instrucoes.py` (regras), `novojogador.py` (personalização), `tema.py` (seleção de tema) e `final.py` (resultados).

3. **Núcleo de Jogo (Game Loop):** Representa o ambiente da partida e o tabuleiro. É composto pelos módulos específicos de tema (`JogoMario.py`, `JogoPokemon.py` e `JogoZelda.py`), que integram a IA (o bot) e gerenciam a jogabilidade em turnos.

2.1.2 Metodologia de Implementação

A escolha pela biblioteca `Pygame` foi fundamental por ser uma ferramenta madura, multiplataforma e de código aberto, amplamente documentada e recomendada para o desenvolvimento rápido de jogos 2D em Python (PYGAME, 2024).

A implementação seguiu o paradigma de programação orientada a eventos e o padrão de *Game Loop* (Loop de Jogo):

- **Loop Principal:** O núcleo de cada módulo de jogo (`Jogo.py`) é baseado em um loop `while True`, que é o cerne da arquitetura de jogos e garante a execução contínua da aplicação. Este loop executa três funções primárias em sequência: a leitura e o tratamento de eventos (como cliques do mouse) através de `pygame.event.get()`, a atualização do estado lógico do jogo (cálculo de pontuação, verificação de pares) e, por fim, a renderização gráfica da nova cena na tela com `pygame.display.flip()`.

- **Gestão de Turnos:** O fluxo do jogo é rigidamente controlado pela variável booleana `turno_jogador`. Esta variável funciona como um mecanismo de chaveamento entre o jogador humano e o agente autônomo. Quando `turno_jogador` é `True`, o loop principal aguarda a entrada do usuário, especificamente o evento `MOUSEBUTTONDOWN`, para que a jogada seja processada. Por outro lado, quando a variável é `False`, o jogo entra no modo de turno da IA, é imposto um atraso de 1 segundo com `pygame.time.wait(1000)` para simular o "pensamento" do bot, seguido da chamada à sua função de decisão `bot.escolher_jogada()`. Este mecanismo evita respostas instantâneas e torna a experiência mais natural para o jogador humano.

2.2 IMPLEMENTAÇÃO DA LÓGICA DE JOGO E IA

O componente central de IA do projeto foi implementado na classe `BotMemoria`, e seu funcionamento detalhado é dividido em três níveis de implementação.

2.2.1 Estrutura de Dados e Limitação de Memória

O componente central de IA do projeto foi implementado na classe `BotMemoria`. A memória do bot é gerenciada por um dicionário Python armazenado na variável `self.memoria`, no qual cada ID de carta (chave) está associado a uma lista de posições no tabuleiro.

A limitação intencional é definida pelo parâmetro `self.tamanho_memoria = 4`. Este limite representa a capacidade total da memória do agente e é essencial para impedir que o bot seja infalível, garantindo um adversário mais equilibrado e realista.

2.2.2 Mecanismos de Aprendizado e Esquecimento

O funcionamento da memória é dinâmico, baseado em dois processos fundamentais:

- **Aprendizado (`atualizar_memoria()`):** Função chamada toda vez que uma carta é virada, armazenando a posição da carta de acordo com o seu ID.
- **Esquecimento por Limitação:** Quando o limite `tamanho_memoria` é ultrapassado, o item mais antigo da lista de posições é removido com `.pop(0)`, simulando uma memória de curto prazo.
- **Esquecimento Intencional (`remover_par_da_memoria()`):** Quando um par é encontrado, o ID correspondente é removido da memória.

2.2.3 *Lógica Hierárquica de Decisão (Prioridades)*

A função `escolher_jogada()` constitui o núcleo da IA, implementando uma lógica hierárquica baseada em prioridades:

- **Prioridade 1 — Memória contém um par completo:** O bot seleciona diretamente as duas posições conhecidas, garantindo o ponto.
- **Prioridade 2 — Memória contém apenas uma carta única:** O bot escolhe a carta conhecida e tenta uma segunda carta aleatória.
- **Prioridade 3 — Nenhuma das condições anteriores é satisfeita:** O bot realiza uma jogada completamente aleatória.

O fluxo completo dessa lógica é representado no fluxograma apresentado na próxima seção.

Figura 1: Fluxograma do algoritmo de tomada de decisão do agente.

2.3 IMPLEMENTAÇÃO DA INTERFACE GRÁFICA E FLUXO DE NAVEGAÇÃO

A interface gráfica (GUI) foi desenvolvida com o objetivo de oferecer uma experiência intuitiva e visualmente atrativa, utilizando o estilo *pixel art* para os *assets* do jogo. Essa escolha estética foi fundamental para garantir a coerência temática com os universos retrô de Super Mario, Pokémon e Zelda, criando uma ambientação lúdica que remete aos clássicos dos videogames.

O desenvolvimento gráfico baseou-se na biblioteca `Pygame`, que forneceu os métodos essenciais para o carregamento e tratamento eficiente de imagens (`pygame.image.load()`) e para a adaptação responsiva dos elementos visuais. O uso de `pygame.transform.smoothscale()` permitiu que os *assets* e o tabuleiro se ajustassem dinamicamente às dimensões da tela do usuário.

O gerenciamento de fontes exclusivas (como `arcade_gamer.ttf`) e a aplicação de cores temáticas desempenharam um papel crucial na construção da identidade visual, assegurando a clara separação entre os elementos de jogabilidade e a estética da interface.

A navegação entre as telas foi projetada de forma linear e simples, priorizando tanto a usabilidade quanto a imersão do jogador.

2.3.1 Telas de Configuração Inicial

O fluxo de navegação garante a correta captura das preferências do jogador e a transição fluida entre as etapas de configuração da partida.

- **Menu Principal e Instruções:** O módulo `Menu.py` carrega a tela inicial e aciona `instrucoes.py` ao clicar no botão `Imagens/BotaoStart.png`. A Figura 2 apresenta o Menu Principal e a figura 3 apresenta as Instruções.

Figura 2: Tela inicial.

Figura 3: Tela de instruções.

- **Tela de Novo Jogador:** O módulo `novojogador.py` permite a personalização. O usuário insere seu nome e seleciona seu avatar. A Figura 4 mostra a tela de Cadastro e Seleção de Avatar.

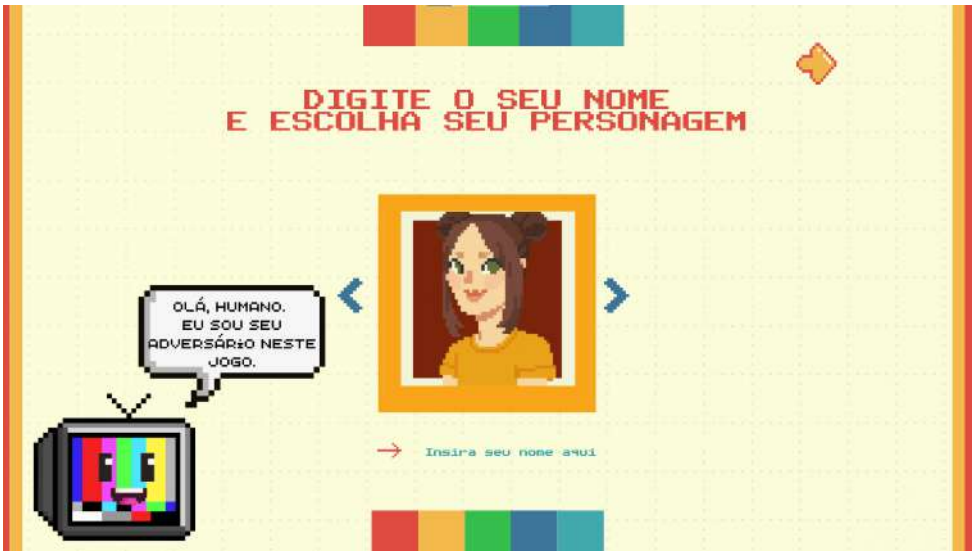

Figura 4: Tela de cadastro e seleção de avatar.

2.3.2 *Tela de Seleção de Tema*

A seção da Interface Gráfica responsável pela escolha temática é gerenciada pelo módulo `tema.py`. Essa tela apresenta visualmente ao jogador os três universos disponíveis: Super Mario Bros, Pokémon e The Legend of Zelda, reforçando o estilo *pixel art* da aplicação. A interação é realizada de forma intuitiva por meio de três elementos visuais distintos, cada um representando uma categoria temática do jogo.

Internamente, o módulo `tema.py` é responsável por carregar e ajustar a escala desses *assets*, garantindo que o layout permaneça coeso e proporcional em diferentes tamanhos de tela. A seleção é acionada a partir do evento de clique (`MOUSEBUTTONDOWN`), que atua como ponto de divergência no fluxo da aplicação, direcionando o jogador para o módulo de jogo correspondente (por exemplo, `JogoMario.py`). Esse módulo, por sua vez, carrega a lógica da partida e o conjunto de cartas temáticas relativas à escolha realizada.

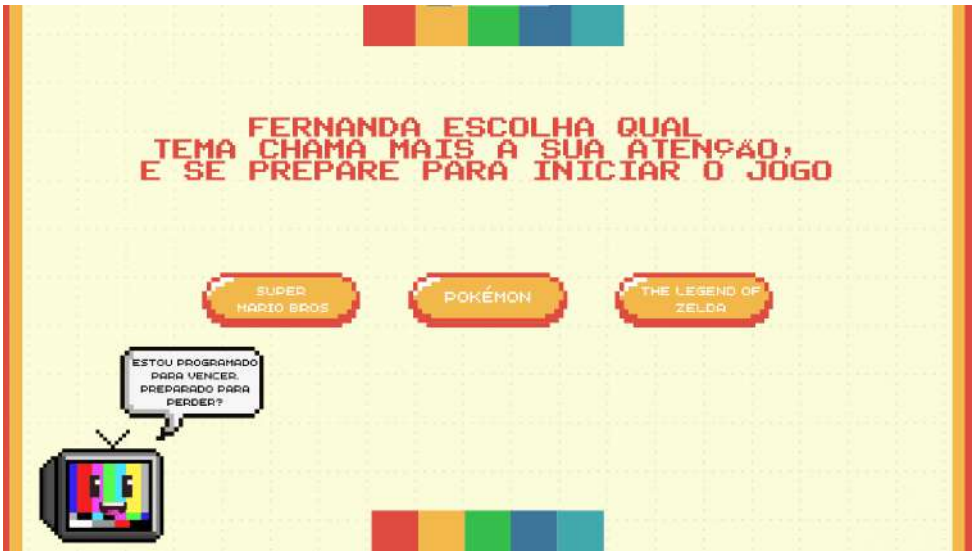

Figura 5: Interface de seleção de tema.

2.3.3 *Interface de Jogo e Conclusão*

Os módulos de jogo (por exemplo, `JogoMario.py`) contêm a lógica principal de *gameplay*. Para iniciar a partida, cada módulo é responsável por carregar dinamicamente seis pares de cartas temáticas (totalizando 12 *assets* visuais). A imprevisibilidade do tabuleiro em cada nova sessão é garantida pela aplicação da função `random.shuffle` à lista de posições das cartas, o que randomiza seu arranjo no *grid*.

Durante a renderização, a interface compõe o tabuleiro utilizando o verso da carta específico de cada tema (por exemplo, o verso utilizado no módulo `JogoMario.py`) posicionado sobre o fundo padronizado do jogo. Esse processo assegura a coerência visual e mantém o estilo temático da interface ao longo de toda a experiência de jogo.

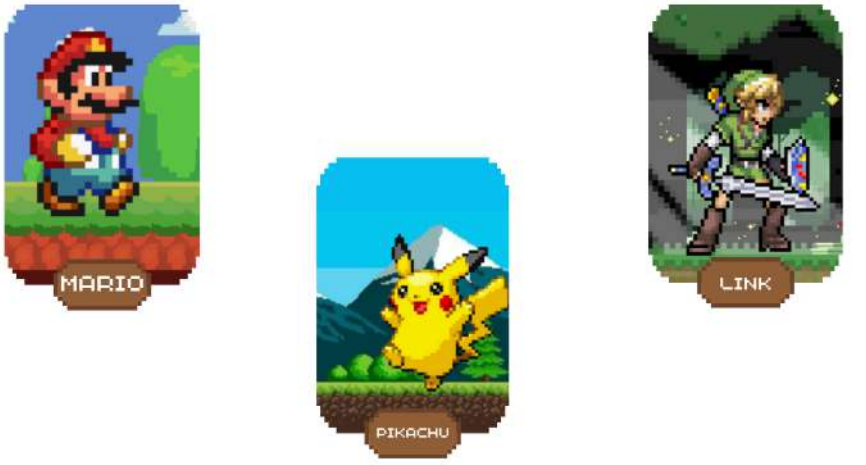

Figura 6: Identidade Visual e Protagonistas dos Três Universos Temáticos (Mario, Pokémon e Zelda)

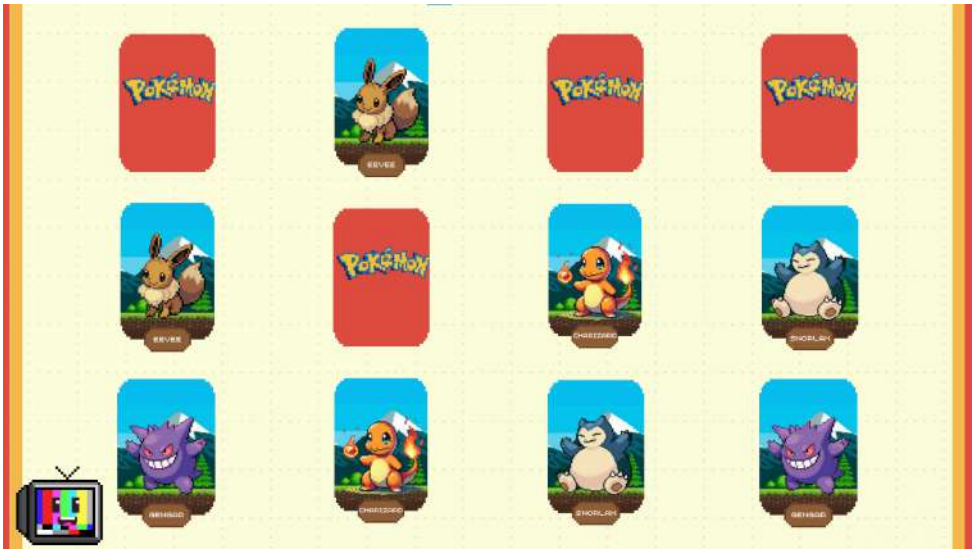

Figura 7: Interface de Partida com Tabuleiro e Cartas Reveladas (Tema Pokémon).

O ciclo de jogo se encerra quando todas as cartas foram reveladas e o tabuleiro está completo. Nesse momento, o fluxo é encaminhado ao módulo `final.py`, que compara as variáveis `pares_jogador` e `pares_bot` para determinar o resultado da partida.

O módulo então carrega a tela correspondente a um dos três estados possíveis:

- **Vitória:** exibida quando a pontuação do jogador é superior.
- **Derrota:** exibida quando o bot obtém a maior pontuação.
- **Empate:** exibida quando as pontuações são iguais.

Figura 8: Vitória do Jogador.

Figura 9: Empate na Partida

Figura 10: Vitória do Bot (Derrota do Jogador)

2.4 Resultados e Discussão

A validação do sistema desenvolvido foi realizada em duas frentes principais: a verificação funcional da interface e do fluxo de jogo, e a análise comportamental do agente computacional (bot). Os testes práticos demonstraram que a arquitetura modular adotada permitiu uma integração eficiente entre a lógica de decisão e a interface gráfica.

2.4.1 Validação Funcional e de Interface

A validação funcional, realizada através de testes de execução, confirmou a estabilidade e a coerência do fluxo de navegação proposto. Foi verificado o trânsito correto e estável por todas as etapas de configuração da partida, desde a tela inicial (menu.py) até o carregamento do ambiente de gameplay nos módulos específicos de jogo.

A interface gráfica demonstrou ser coesa, com todos os assets carregados corretamente e ajustados dinamicamente às dimensões da tela, mantendo a identidade visual pixel art dos temas. A correta captura e transmissão dos dados de personalização (nome do usuário e avatar) entre os módulos também foi confirmada, atestando a eficácia da comunicação entre os componentes da arquitetura modular. O sistema de áudio também manteve-se estável durante toda a sessão, contribuindo para a ambientação lúdica.

2.4.2 Análise Comportamental do Agente (Bot)

O foco central da análise recaiu sobre a validação comportamental do agente computacional. A metodologia de verificação da lógica interna utilizou técnicas de teste de caixa-branca (whitebox testing). Durante a execução dos módulos de jogo, o estado interno da memória do bot foi monitorado em tempo real através de saídas no console, permitindo confirmar que as funções de atualização e esquecimento estavam operando corretamente a cada jogada.

- **Testes da Lógica de Prioridades:** A partir das partidas de teste e da análise via console, confirmou-se que a lógica hierárquica de decisão do bot funciona conforme o planejado, demonstrando domínio progressivo sobre as informações armazenadas.
  - **Prioridade 1 (Jogada Certa):** Em cenários em que um par de cartas já havia sido revelado e registrado na memória, o bot executou consistentemente a jogada correta. Isso validou o funcionamento adequado da Prioridade 1, garantindo que pares conhecidos fossem sempre aproveitados.
  - **Prioridade 2 (Risco Inteligente):** Nos casos em que havia apenas informação parcial, a Prioridade 2 foi acionada. Nessa situação, o bot selecionava uma carta conhecida e arriscava a segunda jogada em uma carta ainda não revelada, simulando um comportamento de busca estratégica e exploração inteligente.
- **Balanceamento e Limite de Memória:** A restrição definida pelo parâmetro `tamanho_memoria = 4` mostrou-se o mecanismo de balanceamento mais eficiente. Observou-se que o bot "esquecia" cartas reveladas após muitos turnos quando novas informações eram inseridas, o que impedia que o agente se tornasse invencível ou artificialmente perfeito. Essa limitação promoveu o equilíbrio desejado entre desafio e realismo, oferecendo uma experiência mais justa e competitiva para o jogador humano.

2.4.3 Competitividade e Gestão de Estados Finais

Os testes de competitividade registraram cenários variados de conclusão de partida. O módulo de conclusão (`final.py`) validou a competitividade do sistema: foram registrados cenários de vitória do jogador, vitória do bot ("O melhor bot"), e empates, com a exibição correta das telas de resultado correspondentes. Isso comprovou que o agente desenvolvido é capaz de competir efetivamente contra jogadores humanos, e que o sistema de gestão de

estados (vitória, derrota, empate) respondeu corretamente em todas as simulações, validando a robustez do software.

3 CONCLUSÃO

Este trabalho apresentou o desenvolvimento completo de um Jogo da Memória digital com um agente computacional integrado, cumprindo o objetivo principal de fornecer uma ferramenta lúdica e tecnológica para o contexto do Projeto LACIA. A implementação da solução em Python, utilizando a biblioteca Pygame, mostrou-se uma escolha adequada, permitindo a criação de uma interface gráfica responsiva e modular que suporta múltiplos temas (Super Mario, Pokémon e Zelda).

Os objetivos específicos propostos foram plenamente alcançados. A modelagem da lógica do jogo e o sistema de turnos garantiram uma jogabilidade fluida e coerente com as regras clássicas. A principal contribuição técnica, o desenvolvimento do bot com memória simulada, provou ser eficaz. A estratégia de limitar a memória do agente a quatro posições (`tamanho_memoria = 4`) atingiu o equilíbrio desejado entre desafio e realismo, evitando que o computador fosse um oponente infalível e criando uma experiência de disputa justa para o jogador humano.

Os testes funcionais e comportamentais validaram a robustez do software. O agente demonstrou capacidade de tomar decisões racionais baseadas em prioridades — formando pares conhecidos ou arriscando jogadas exploratórias — e o sistema de gestão de estados (vitória, derrota, empate) respondeu corretamente em todas as simulações. Além disso, a arquitetura modular do código assegura que o projeto seja extensível e de fácil manutenção.

Como contribuição acadêmica e social, este software serve agora como um recurso didático para o LACIA, podendo ser utilizado tanto para o entretenimento quanto como objeto de estudo para alunos iniciantes em programação e inteligência artificial.

Para trabalhos futuros, sugere-se a implementação de níveis de dificuldade ajustáveis, onde o parâmetro de memória do bot possa variar dinamicamente (ex: níveis "Fácil", "Médio"e "Difícil"). Outra possibilidade de expansão seria a aplicação de técnicas de Aprendizado de Máquina (Machine Learning) para que o bot possa adaptar sua estratégia com base no comportamento do jogador ao longo de múltiplas partidas.

Referências

ALVES, L.; COUTINHO, I. d. J. (Ed.). *Jogos digitais e aprendizagem: Fundamentos para uma prática baseada em evidências*. [S.l.]: Papirus Editora, 2020.

BARBOSA, E. F.; MOURA, D. Guimarães de. Metodologias ativas de aprendizagem na educação profissional e tecnológica. *Boletim Técnico Do Senac*, v. 39, n. 2, p. 48–67, 2013.

BORGES, L. E. *Python para desenvolvedores: Aborda Python 3.3*. [S.l.]: Novatec Editora, 2014.

FILHO, A. R. G.; ANDRADE, K. de; SOUZA, R. de. Inteligência artificial em jogos digitais. *Interface Tecnológica*, v. 19, n. 2, p. 342–354, 2022. Disponível em: <https://revista.fatectq.edu.br/interfacetecnologica/article/view/1246>.

MUNÁRRIZ, L. Á. *Fundamentos de inteligencia artificial*. Espanha: Universidad de Murcia, 1994.

PYGAME. *Pygame Documentação Oficial*. 2024. Disponível em: <https://www.pygame.org/docs/>. Acesso em: 15 nov. 2024.

REBOUÇAS, A. et al. Aprendendo a ensinar programação combinando jogos e python. *Interface Tecnológica*, v. 10, n. 1, p. 270–281, 2013. Disponível em: <https://revista.fatectq.edu.br/interfacetecnologica/article/view/1546>.

SWEIGART, A. *Making games with Python & Pygame*. [S.l.: s.n.], 2012. Disponível em: <https://inventwithpython.com/makinggames.pdf>. Acesso em: 15 nov. 2024.