PUC-SP

Pontifícia Universidade Católica de São Paulo

Especialização em Engenharia de Software

Juliana Felipe Cândido

Uma proposta de Agente com capacidade de aprendizagem

São Paulo

2023

Juliana Felipe Cândido

Uma proposta de Agente com capacidade de aprendizagem

Trabalho de Conclusão de Curso apresentado à banca examinadora da Pontifícia Universidade Católica de São Paulo, como exigência parcial para obtenção do título de Especialista em Engenharia de Software, sob a orientação do(a) prof., dr. Silvio Luiz Stanzani.

São Paulo

2023

Banca Examinadora

_____

_____

_____

_____

_____

À comunidade da Pontifícia Universidade Católica de São Paulo pelo apoio permanente.

AGRADECIMENTOS

Agradeço ao meu orientador, por todo conhecimento compartilhado nessa jornada.
A todos os meus professores, que agregaram e me motivaram ir além do que eu esperava nesse curso.

Aos meus pais, que sempre me incentivaram a nunca parar de estudar, e por mais difícil que seja, sempre correr atrás dos meus sonhos.

Boa sorte é o que acontece quando a oportunidade se encontra com o planejamento.

RESUMO

CÂNDIDO, Juliana. **Uma proposta de Agente com capacidade de aprendizagem.**

A área de Agentes vêm sendo impulsionada pela comunidade de Inteligência Artificial, as técnicas de desenvolvimento desses Agentes autônomos, ou inteligentes, bem como aprendizado de máquina e visão computacional, têm beneficiado muito a área de jogos digitais.

Jogos que são realizados em turnos, tais damas, xadrez, entre outros, necessitam de habilidades como: percepção de ambiente, execução de passos de maneira ordenada e estratégica, bem como a avaliação da qualidade dessas ações.

O objetivo deste trabalho é propor um Agente capaz de jogar Angry Birds, através de técnicas como visão computacional, aprendizado de máquina e estratégia implementada por Agentes para resolver todos os níveis do jogo.

Os experimentos foram executados em um ambiente virtual no Google Colab, e a etapa de treinamento do modelo utilizando inteligência artificial foi executada pelo algoritmo YOLO (*You Only Look Once*).

Palavras-chave: Agente Inteligente; Planejamento Automatizado; Aprendizado de Máquina; Agente para Jogos Digitais.

ABSTRACT

CÂNDIDO, Juliana. **A proposal for an Agent with learning capacity.**

The area of agents has been driven by the artificial intelligence community. The techniques to develop these autonomous or intelligent agents, as well as machine learning and computer vision, have greatly benefited the area of digital games.

Turn games such as checkers, and chess, among others, require skills such as perception of the environment, execution of steps in an orderly and strategic way, and the evaluation of the quality of these actions.

The objective of this work is to propose an agent capable of playing Angry Birds, with techniques such as computer vision, machine learning, and strategy implemented by agents to solve all levels of the game.

The experiments were performed in a virtual environment of Google Colab, and the model training step using artificial intelligence was performed by the YOLO (You Only Look Once) algorithm.

Keywords: Intelligent Agent; Automated Planning; Machine Learning; Agent for Digital Games.

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

AI	Artificial Intelligence
NCP	Non Playable Character
YOLO	You Only Look Once
R-CNN	Region-based Convolutional Neural Network
CSV	Comma-separated Values
HTTPS	HyperText Transfer Protocol Secure
GPU	Graphics Processing Unit
DRY	Don't Repeat Yourself
API	Application Programming Interface

SUMÁRIO

1 Introdução

A inteligência artificial refere-se a máquinas e sistemas que mimetizam a inteligência humana a fim de executar tarefas, podendo se aprimorar de modo iterativo com base em informações coletadas [1]. As técnicas de AI (*Artificial Intelligence*) podem ser explorada de diversas formas:

- Os assistentes inteligentes utilizam AI para realizar análises de grandes conjuntos de dados de linguagem natural (textos livres) para melhorar a programação.
- Chatbots que utilizam AI para fornecer respostas cada vez mais rápidas aos clientes entendendo melhor seus problemas.
- Algoritmos de recomendações que podem fornecer recomendações baseados nos hábitos dos usuários.

A AI está mais relacionada à capacidade e processo de pensamento poderoso, bem como a análise de dados do que algum formato ou função em particular. Seu objetivo é melhorar significativamente as contribuições e habilidades humanas, fazendo dela um ativo de negócio muito valioso nos dias de hoje.

Nos últimos anos, a área de jogos têm crescido muito, em especial o desenvolvimento de AI para a criação de componentes com capacidade cada vez mais avançada para resolução de jogos complexos, com comportamentos mais próximos de um humano. Para o propósito deste trabalho será explorado a capacidade de um Agente aprender como jogar o jogo de videogame **Angry Birds** [2]. O jogo requer resolver obstáculos, raciocinar sobre a consequência de ações, atuar mesmo com observabilidade parcial e sem um modelo completo do ambiente, entre outros desafios. Tudo isso em um ambiente virtual facilmente **observável** (do ponto de vista do humano avaliando a técnica de AI), **modificável** e de onde **métricas de desempenho** podem ser extraídas facilmente (pontuação do jogo).

Ainda sobre o jogo, podemos destacar que: trata-se de um jogo de estratégia, em que o Agente deve executar o disparo de projéteis (pássaros) em alvos definidos (porcos). Estes alvos estão protegidos por estruturas, e deve-se destruí-las para

atingir os alvos. Parece uma tarefa simples, bem direta. Todavia, essa tarefa se torna complexa devido à dificuldade de estimar as consequências de cada disparo, que é efetuado através da análise de imagens, reconhecimento de objetos (através de um sistema denominado **YOLO** [3]).

Elementos do ambiente do Agente:

- Porcos (podem apresentar variação de tamanho)
- Pássaros
- Blocos (podem apresentar variação de resistência aos danos)
- Estilingue (local inicial do lançamento)

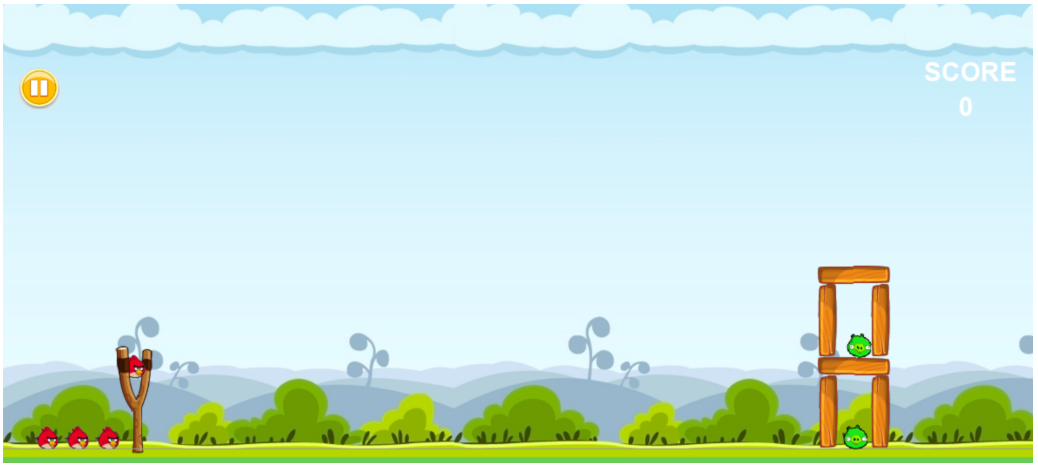

Figura 1: Exemplo do ambiente do Agente

O desenvolvimento desse Agente será válido para a aplicação de técnicas de AI em aplicações no mundo real. O presente documento tem, portanto, o objetivo de demonstrar detalhes de uma estratégia utilizando heurística, bem como a utilização de ferramentas para processamento de imagem e aprendizado de máquina.

1.1 Objetivo Geral

Desenvolver um algoritmo para jogar Angry Birds de modo autônomo. Aplicar técnicas de processamento de imagem, e aprendizado de máquina para atingir esse objetivo.

1.2 Objetivos Específicos

Esse trabalho **não visa alcançar um desempenho melhor do que o humano**, isso ainda é um problema em aberto e depende de futuras pesquisas. A estratégia proposta deverá idealmente:

- O Agente deve melhorar seu desempenho baseado em informações de níveis anteriores e jogadas similares.

1.3 Escopo do Projeto

O escopo desse projeto limita-se apenas ao domínio do Angry Birds, além de estar limitado a versão específica do jogo. Vale ressaltar que os testes do Agente foram executados em um computador local, já o treinamento do modelo que o Agente utiliza para reconhecer os objetos foi efetuado em uma máquina remota no ecossistema do do **Google Colab** [4].

O Agente sempre receberá um nível aleatório para resolver com um conjunto limitado de elementos gráficos.

A cada turno realizado será observado se o Agente ganhou ou não aquela partida e no caso de não ter ganhado, o Agente deverá reiniciar aquele mesmo nível, com o propósito de aprender com suas jogadas passadas.

Foi determinado as seguintes regras:

- Ao ganhar o Agente deve seguir para o próximo nível aleatório.
- Ao perder o Agente poderá tentar o mesmo nível 3 vezes.
- Ao perder 3 vezes seguidas o Agente deverá seguir para o próximo nível aleatório.

1.4 Organização do Texto

Esta monografia está organizada da seguinte maneira: a introdução será apresentada no primeiro capítulo e contemplará os aspectos gerais do presente trabalho, bem como seus objetivos e escopo determinado, de modo a resumir os capítulos que virão a seguir.

Por seguinte teremos a revisão bibliográfica, que proverá uma base teórica, e irá explorar trabalhos relacionados a este.

O próximo capítulo tratará do método de desenvolvimento de todo o projeto, e irá analisar todas as etapas necessárias para construção do software desenvolvido no decorrer deste trabalho. Este, será seguido pelo capítulo de arquitetura de software e tecnologias e seus aspectos de implementação, onde aprofundaremos nos detalhes de cada tipo de visão, sejam estas: estruturais, comportamentais ou de implementação.

No capítulo de testes e análise de resultados serão apresentados cada verificação e suas constatações baseados em um sistema de pontos, onde poderemos entender a conclusão dos experimentos.

E por fim, teremos as conclusões deste projeto e considerações finais, que irão validar tudo o que foi explorado anteriormente.

2 Revisão Bibliográfica

Nesta seção serão abordados conceitos importantes para o desenvolvimento deste trabalho, através de uma fundação teórica, que nos ajudará a entender os elementos necessários para desenvolver um Agente capaz de jogar jogos digitais.

2.1 Agentes e jogos digitais

A correlação entre Agentes e jogos digitais é bastante significativa. Podemos descrever um Agente inteligente como um programa de computador, que é capaz de

tomar decisões e agir autonomamente com base em informações que recebe do ambiente [1].

Em jogos de videogame, por exemplo, os Agentes são usados para criar personagens não-jogáveis identificados como **NPC** (*Non Playable Character*) que interagem com o jogador. Eles precisam ser capazes de tomar decisões, aprender com suas experiências e se adaptar a mudanças no ambiente do jogo. Essas são habilidades importantes que também são características de um Agente inteligente [5].

Observando a evolução dos jogos digitais e da inteligência artificial percebemos que os jogos começaram de modo simples, como *"Pong"*, por exemplo, que tinha gráficos básicos e jogabilidade limitada. No entanto, à medida que a tecnologia avançou, esses jogos se tornaram cada vez mais complexos, com gráficos e jogabilidade mais sofisticados. E assim, a necessidade de Agentes para controlar NPCs aumentou [5][6].

Nos últimos anos, a inteligência artificial tem desempenhado um papel cada vez mais importante na área de jogos. Algoritmos de aprendizado de máquina e redes neurais são usados para criar NPCs mais inteligentes e prever as ações do jogador.

Além disso, algoritmos de inteligência artificial ajudam a melhorar a jogabilidade, criar desafios mais interessantes e personalizar a experiência do jogador. Um exemplo disso é a realidade virtual e aumentada, que estão mudando a maneira como jogamos videogames [5][6]. Os jogos de realidade virtual permitem que os jogadores experimentem mundos virtuais totalmente imersivos e interajam com NPCs mais realistas. A inteligência artificial é uma parte essencial para tornar essas experiências de jogo mais realistas e convincentes.

Em resumo, a evolução dos videogames e da inteligência artificial estão intimamente ligadas. Os jogos de videogame estão se tornando cada vez mais complexos e exigem Agentes cada vez mais inteligentes para controlar NPCs e melhorar a jogabilidade.

2.2 Aprendizado por transferência

Os humanos são muito bons em transferir conhecimento durante o trabalho. Isso significa que sempre que encontramos um novo problema ou tarefa, nós o identificamos e utilizamos nosso conhecimento relevante de nossa experiência de aprendizado anterior. Isso torna nosso trabalho fácil e rápido de ser concluído. Por exemplo, se você sabe andar de bicicleta e lhe pedem para dirigir uma motocicleta, o que você nunca fez antes. Nesse caso, nossa experiência com a bicicleta entrará em jogo e executamos tarefas como equilibrar a bicicleta, dirigir, etc. Isso tornará as coisas mais fáceis em comparação com um iniciante completo. Essas tendências são muito úteis na vida real. Porque nos torna mais perfeitos e nos ajuda a ganhar mais experiência.

Seguindo a mesma abordagem, o termo aprendizado de transferência foi introduzido no campo do aprendizado de máquina [7][8]. Essa abordagem consiste em pegar o conhecimento adquirido em uma tarefa e usar isso para resolver o problema na tarefa objetivo associada. Embora a maior parte do aprendizado de máquina seja projetada para lidar com uma única tarefa, o desenvolvimento de algoritmos para facilitar o aprendizado de transferência é um tópico de interesse contínuo na comunidade de aprendizado de máquina.

Existem três tipos de transferência de aprendizado: positiva, negativa e nula. A transferência positiva ocorre quando a aprendizagem em uma situação melhora o desempenho em outra situação. Por sua vez, a transferência negativa ocorre quando a aprendizagem em uma situação prejudica o desempenho em outra situação. Já a transferência nula ocorre quando não há efeito da aprendizagem em uma situação sobre o desempenho em outra situação [9].

Um exemplo de transferência positiva é quando um jogador aprende a usar estratégias de gerenciamento de recursos em um jogo de estratégia e, em seguida, usa essas mesmas estratégias em um jogo de simulação empresarial. Já um exemplo de transferência negativa é quando um jogador aprende a usar um controle de jogo específico em um jogo e, em seguida, tem dificuldade em usar um controle diferente em outro jogo. Por fim, um exemplo de transferência nula é quando um

jogador aprende a jogar um jogo de futebol e, em seguida, joga um jogo de corrida que não exige as mesmas habilidades.

Em resumo, a transferência de aprendizado é a capacidade de usar o que foi aprendido em uma determinada situação para resolver problemas em outras situações semelhantes ou diferentes [7][8].

2.3 Visão computacional

A visão computacional é um campo da ciência da computação e da inteligência artificial que visa analisar, interpretar e extrair informações relevantes de imagens e/ou vídeos para tomar decisões ou gerar dados relevantes para futuras aplicações [10]. A ideia é detectar fenômenos da mesma forma que a visão humana, na medida em que somos capazes de detectar objetos, entender a distância e entender o que está no ambiente.

Atualmente, podemos notar o uso de inteligência artificial e visão computacional em diversos domínios, desde aplicativos bancários de rotina, até a detecção de doenças por meio de imagens, e também em carros autônomos.

Um exemplo de aplicação de visão computacional é o reconhecimento facial. Os algoritmos de visão computacional são capazes de identificar as características faciais de uma pessoa em uma imagem ou vídeo e, em seguida, compará-las com outras imagens em um banco de dados para identificar a pessoa. Esse tipo de tecnologia é amplamente utilizado em segurança, como em sistemas de controle de acesso, e em redes sociais, como o reconhecimento de amigos em fotos.

Outro exemplo é a detecção de objetos. Algoritmos de visão computacional podem identificar e rastrear objetos em tempo real, como carros em uma estrada ou pessoas em um shopping center. Isso é usado em sistemas de monitoramento de segurança, navegação de veículos autônomos e em jogos de realidade aumentada [10].

A visão computacional também é usada em diagnóstico médico, onde algoritmos são capazes de analisar imagens médicas, como radiografias e tomografias, para detectar doenças e condições.

Em resumo, a visão computacional é uma área da inteligência artificial que permite que os computadores percebam e entendam o mundo visual ao seu redor. Sua aplicação é vasta e muito relevante para o mundo de hoje.

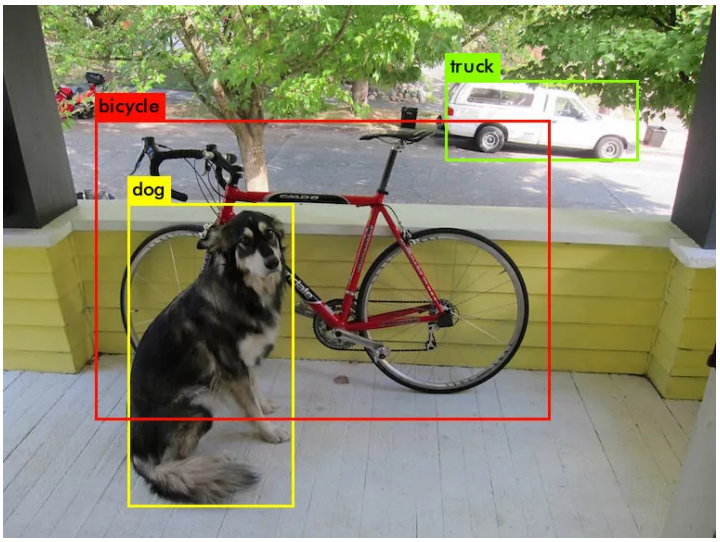

Figura 2: Exemplo de objetos sendo detectados pelo YOLO

2.4 Fundamentação Teórica

Diversas técnicas e ferramentas foram utilizadas no desenvolvimento do Agente e do modelo computacional que foi treinado para reconhecer os objetos do jogo. Inicialmente, para o treinamento do modelo foi utilizado uma tecnologia denominada YOLO**,** que é um método de detecção de objetos de passada única (*single pass*) que utiliza uma rede neural convolucional como extrator de características (*features*).

Diferente de algoritmos anteriores de detecção de objetos, como **R-CNN** (*Region-based Convolutional Neural Network*) ou **Faster R-CNN** [11], ele apenas precisa olhar pela imagem uma única vez para enviar para a rede neural, por isso ele recebe esse nome.

E devido a essa característica, o YOLO foi capaz de conseguir uma velocidade na detecção muito maior do que as técnicas concorrentes, sem perder em acurácia.

A ilustração abaixo representa graficamente suas características:

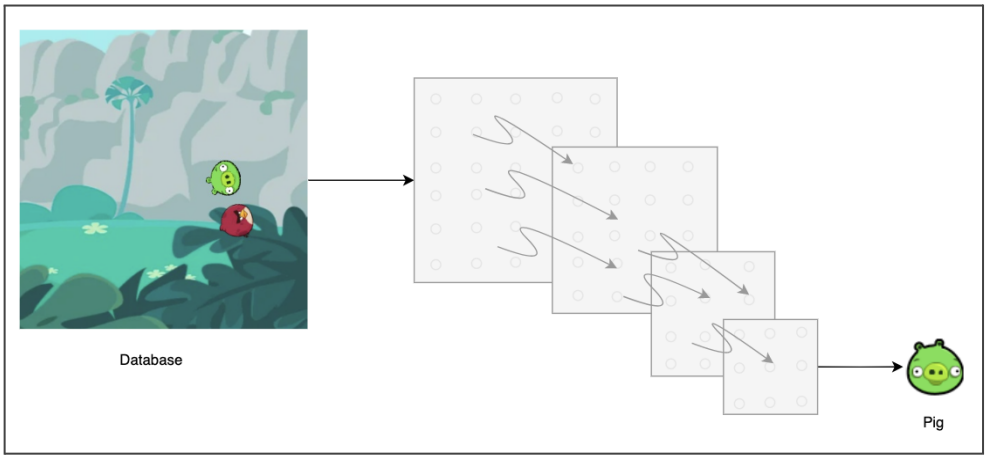

Figura 3: Representação do algoritmo YOLO

Atualmente, ele é o estado da arte em sistemas de reconhecimento de objetos em tempo real, apesar de existir um conflito entre velocidade e assertividade.

É uma ferramenta de código aberto, ou seja, tudo nesta tecnologia (o código-fonte, a arquitetura da rede neural, os pesos com os quais esta rede é executada e os conjuntos de dados utilizados para treinar a rede) é livre e pode ser usado por qualquer um.

O YOLO por padrão já foi treinado com o conjunto de dados do **MS COCO** [12], que possui 80 classes diferentes. E os pesos (parâmetros de treinamento) devem ser obtidos separadamente.

2.5 Trabalhos Relacionados

O desenvolvimento de Agentes para jogos digitais e treinamento de modelos de visão computacional são tópicos bastante discutidos hoje em dia. Como foi citado anteriormente, a indústria de jogos eletrônicos tem uma relação cada vez mais forte com a inteligência artificial e os Agentes inteligentes. Isso se deve, em grande parte, à necessidade de criar experiências de jogo mais realistas e interativas, que possam

adaptar-se ao comportamento dos jogadores e proporcionar desafios cada vez mais interessantes.

Um exemplo disso é o jogo *"The Last of Us Part II"*. Este jogo de ação e aventura desenvolvido pela *Naughty Dog* [13] utiliza inteligência artificial para criar NPCs que se comportam de forma realista e interativa. Os NPCs são capazes de tomar decisões autônomas com base em um conjunto de regras e algoritmos, criando uma sensação de imersão e tornando a experiência de jogo mais realista.

Ainda como exemplo temos diversos trabalhos relacionados ao tema no mundo acadêmico; sempre focando em um determinado aspecto.

Exemplos de AI aplicadas a jogos:

2.6 Inteligência artificial em jogos digitais

Comentamos anteriormente sobre como a inteligência artificial desempenha um papel fundamental no desenvolvimento de jogos digitais, e como é frequentemente utilizada para aprimorar a experiência do jogador, sua imersão no jogo e desafiar suas habilidades de formular estratégias que sejam melhores do que os Agentes, ou NPCs.

No caso específico do jogo Angry Birds, podemos evidenciar várias técnicas de AI, tais quais: árvores de comportamento, máquinas de estados finitos e algoritmos genéticos [14][15]. Além disso, vale também destacar a importância de técnicas como redes neurais artificiais e aprendizado de máquina, que foram utilizadas no presente trabalho.

Os Agentes são entidades virtuais que interagem e interpretam o ambiente, no caso do jogo, interagem e interpretam seu cenário, objetos, obstáculos, etc. Do ponto de vista do observador, podemos dizer que, no Angry Birds, os pássaros são exemplos de Agentes. Esses Agentes podem ser controlados por algoritmos de AI, que determinam suas ações e comportamentos, por exemplo: o cálculo da trajetória ideal

de lançamento dos pássaros, levando em consideração a posição dos porcos e dos obstáculos [14][15].

Esses algoritmos e estratégias utilizadas pelo Agente são técnicas úteis para encontrar soluções melhores e mais eficientes para finalizar o jogo. Ao aplicar esses algoritmos podemos ajustar automaticamente os parâmetros dos próprios Agentes, como a força do lançamento ou a capacidade de destruição dos pássaros [15]. Dessa forma, os Agentes evoluem ao longo do tempo, tornando-se mais habilidosos e capazes de superar desafios mais difíceis.

Ao que se trata de máquinas de estados finitos, podemos dizer que são modelos computacionais que descrevem o comportamento de um Agente com base em estados e transições entre eles [1]. No Angry Birds, os Agentes podem ser programados com máquinas de estados finitos para definir suas ações em diferentes situações. Por exemplo, um pássaro pode ter um estado de "preparação para o lançamento", seguido por um estado de "voo em direção aos porcos" e, finalmente, um estado de "colisão com os porcos" [15].

Sendo assim, ao utilizar as técnicas citadas acima ao mesmo tempo, podemos assumir erroneamente, que o Agente seria capaz de ter um desempenho melhor do que um humano, afinal, cada camada matemática, de comportamento, de estratégia e de conhecimento foram pensadas e processadas por uma máquina. Todavia, ao longo deste trabalho esta questão e outras mais serão detalhadas, demonstrando o por que este pensamento não está correto. Demonstrando que, apesar do alto desempenho, quais são as limitações encontradas, bem como dificuldades.

2.7 Ferramenta para detecção inteligente de objetos

É possível utilizar a rede neural do YOLO para detectar diversos tipos de objetos em tempo real.

Em um jogo de xadrez, por exemplo, essa técnica permite que o jogo reaja de forma inteligente às ações do jogador, a implementação do algoritmo YOLO, em conjunto a

um ambiente onde o jogador pode controlar um personagem que precisa coletar objetos para avançar no jogo podem ser grandes desafios computacionais [16].

3 Método de Desenvolvimento

O processo de desenvolvimento deste projeto como um todo se deu em algumas etapas. O seu foco principal era utilizar um modelo de visão computacional treinado para identificar objetos (porcos) em uma versão específica do jogo Angry Birds. Mas, indo um pouco além, foram desenvolvidas três estratégias diferentes, para fins de comparação através de um sistema de pontuação, para demonstrar a evolução do Agente no jogo.

Além disso, outro ponto que vale ser destacado, que será mostrado mais adiante, é como a arquitetura influenciou positivamente o desenvolvimento deste software. O baixo acoplamento entre os módulos, possibilita que o módulo de visão e do Agente possam ser substituídos por outros módulos tranquilamente.

Sobre o processo de construção dessa aplicação, temos as seguintes etapas:

3.1 Desenvolvimento de dataset customizado

Antes de iniciar o treinamento do modelo era necessário um dataset, que é o *"input"* de dados utilizado pelo YOLO para reconhecer algum determinado objeto.

Inicialmente, foi desenvolvido um código para gerar imagens aleatórias que continham todos os objetos do jogo, como porcos e pássaros, por exemplo. Essa etapa é muito importante para que o algoritmo do YOLO saiba como classificar corretamente cada objeto.

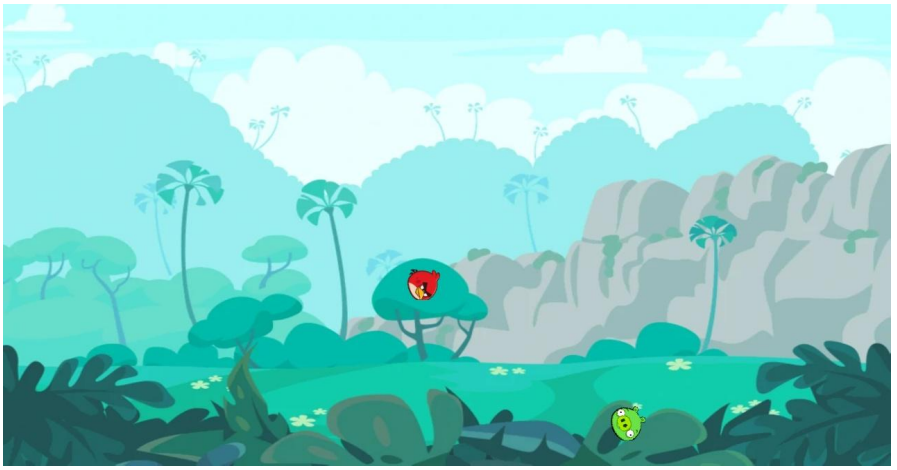

Figura 4: Amostra do conjunto de dados sintéticos

Após o conjunto de dados sintéticos ser criado, foi necessário definir o escopo de detecção de objetos. Sendo assim, foi estabelecido que o YOLO deveria classificar apenas um objeto: o porco. Isso facilita o desenvolvimento do algoritmo e acelera o processo de treinamento do modelo. Como o tempo é uma questão crucial na etapa de treinamento, essa foi uma decisão bem assertiva, que ajudará a acelerar as próximas etapas.

Por fim, um último detalhe foi adicionado. Precisamos mapear corretamente os objetos presentes nas imagens, através de anotações, que nada mais são do que arquivos de texto com a posição de cada objeto de determinada imagem, no plano cartesiano. O arquivo final criado é uma planilha em formato **CSV** (*Comma-separated Values*), que contém o nome do arquivo e os dados gerados pelas anotações (posições dos elementos).

3.2 Treinamento de modelo de visão computacional

Com o arquivo de dados em mãos podemos seguir para o próximo passo: treinar o modelo de visão computacional, utilizando o algoritmo do YOLO.

Esta etapa foi complexa, por que, necessitou de diversos passos para ser executada. Além da fase de treinamento em si, que chegou a durar mais de 18 horas, foi necessário a utilização de hardware específico para isso. O algoritmo do YOLO precisa de uma GPU (*Graphics Processing Unit*) para funcionar corretamente,

o que pode ser muito caro. Então a solução encontrada foi utilizar um ambiente virtual, com esses requisitos, o **Google Colab**.

O Google Colab é um produto do **Google Research** [17], área de pesquisas científicas do Google. Ele permite escrever e executar código **Python** [18] arbitrário pelo navegador e é especialmente adequado para aprendizado de máquina, análise de dados etc. Nesta plataforma foi possível requisitar o uso de GPU, o que viabilizou este projeto.

Mesmo com o ambiente pronto, como não foi utilizado um conjunto de dados já conhecido, foi necessário fazer uma série de modificações para que o algoritmo consumisse o conjunto de dados sintéticos criados, e considerasse as classes definidas (porcos).

As principais etapas para o treinamento do modelo envolveram:

- Atualizar o driver de vídeo.
- Efetuar download de um repositório com o algoritmo de treinamento YOLO.
- Efetuar download dos pesos (parâmetros de treinamento) utilizados pela ferramenta.
- Atualizar todas as referências para os dados gerados na etapa anterior (afinal estamos utilizando um dataset customizado).
- Executar o treinamento.

Parece simples, mas essa etapa foi a que mais sofreu mudanças, e teve problemas no momento de execução. Os motivos, foram diversos, como por exemplo:

- O Google Colab força o servidor a se desconectar depois de algumas horas, prejudicando a etapa de treinamento.
- As etapas de download e atualização de arquivos precisaram ser refeitas diversas vezes, pois os arquivos ficavam hospedados por tempo limitado no ambiente montado.
- A versão de algumas bibliotecas precisou ser alterada.

Todavia, após modificar todos os itens necessários e diminuir o tamanho do conjunto de dados foi possível realizar diversos testes, muito interessantes, que serão apresentados posteriormente.

3.3 Desenvolvimento de código

Inicialmente foi desenvolvido um módulo de *"debugger"*, para explorar a visualização dos elementos na tela. Esse módulo é bastante simples, e seu foco é apenas mostrar a tela do jogo e as variáveis mais importantes, como ângulo, posição inicial do pássaro, entre outras. Isto foi muito importante para poder observar os problemas que poderiam ser encontrados posteriormente, e corrigi-los.

Após o desenvolvimento desse módulo foi criado toda a parte responsável por executar as ações na tela. Nesta fase foram adicionadas bibliotecas como **Numpy** [19] e **OpenCV** [20], que serão mostradas posteriormente. A seguir, foram desenvolvidos os módulos de cálculo de trajetória e visão computacional, que serviram de base para o Agente executar cada jogada, mas neste ponto ainda não existia uma estratégia para o jogo, apenas jogadas aleatórias.

Apesar de a estratégia do Agente em si não ser o foco deste trabalho, foram implementadas cerca de 3 estratégias para o Agente. Visto que, jogadas aleatórias não são tão interessantes para o demonstrativo de resultados. As estratégias são:

- **Estratégia 1:** jogadas aleatórias.
- **Estratégia 2:** jogadas com delimitação de ângulo.
- **Estratégia 3:** jogadas com mira específica no alvo (implementação do modelo treinado).

A cada fase o Agente melhorou seu resultado nas jogadas e após a implementação do modelo já treinado, na terceira fase, foi onde obteve seus melhores resultados.

No final deste projeto ainda foi possível fazer um incremento, onde os dados gerados foram coletados, através de planilhas em formato CSV, e utilizados

posteriormente como parâmetro de qualidade das jogadas. Melhorando ainda mais a performance do Agente. Esta parte será demonstrada na seção de análises e testes.

3.4 Execução de testes

Ao desenvolver cada fase da estratégia do Agente foi executado alguns testes, que se basearam na pontuação do Agente no final de cada nível.

Vale destacar que, a fim de padronizar os testes, foram considerados 10 níveis consecutivos, além de alterações para que o Agente jogasse os mesmos 10 níveis em cada estratégia desenvolvida.

Os resultados destes testes serão explorados posteriormente, todavia, é válido elucidar as etapas para execução dos testes:

- Escolher 10 níveis com dificuldades variadas, porém crescentes.
- Especificar no código a estratégia a ser utilizada.
- Especificar no código o conjunto de níveis a serem jogados.
- Executar todos os níveis em sequência.

Vale ressaltar, que esta última etapa onde executamos o jogo para todos os níveis em sequência deve seguir as regras citadas na seção de escopo do presente trabalho, onde ao perder o Agente pode ter 3 chances de executar aquele mesmo nível. Isto servirá para podermos identificar se alguma estratégia não conseguiu ir até o final do nível, por exemplo. Por fim, vamos guardar o resultado de cada teste em um arquivo em formato CSV, para cada nível jogado, separando por tipo de estratégia.

3.5 Nível de confiança

O nível de confiança em modelos de aprendizado de máquina e visão computacional refere-se à medida de certeza ou confiabilidade que um modelo tem em suas previsões ou inferências. Isto seja, o nível de confiança pode ser associado à detecção e classificação de objetos em uma imagem. No caso deste trabalho,

estamos interessados em classificar itens como porcos e pássaros em um jogo. Podemos então afirmar, que esta é uma métrica importante para avaliar a confiabilidade e a qualidade das previsões feitas por um modelo de AI [21].

Geralmente expresso como uma probabilidade, um modelo pode prever, por exemplo, que uma imagem contém um gato com 80% de confiança. Isso significa que o modelo está 80% certo em sua previsão. Vale ressaltar que esta é uma medida crítica, pois permite que o software compreenda o quão confiáveis são as inferências feitas por este modelo. Ele também pode ser usado para definir limites de confiança para tomar decisões baseadas nas previsões do modelo.

Os Agentes estão relacionados com o nível de confiança no sentido de que eles podem usar essa informação para tomar decisões e agir de acordo com as previsões ou inferências feitas pelo modelo. Por exemplo, um Agente pode ser configurado para tomar uma ação apenas se o nível de confiança do modelo estiver acima de um determinado limite predefinido. Isso permite que o Agente seja mais cauteloso e evite tomar decisões com base em previsões menos confiáveis.

É importante ressaltar que os Agentes também consideram outras informações e contextos relevantes ao tomar decisões, além do nível de confiança do modelo. Isso ajuda a evitar confiar cegamente em previsões com altos níveis de confiança, mesmo quando outros fatores indicam que as previsões podem estar incorretas.

Todavia, é importante observar que o nível de confiança não é uma medida absoluta de precisão. Cada modelo pode ter diferentes níveis de confiança em diferentes situações e podem cometer erros, mesmo com altos níveis de confiança. Portanto, é essencial interpretar e validar as previsões do modelo considerando o contexto e a aplicação específica. O que veremos mais a seguir.

4 Arquitetura do Software

A arquitetura desse software pode ser compreendida em duas partes separadas. A primeira parte, que trata do modelo de aprendizado de máquina, que irá aprender em um ambiente externo (Google Colab), conectado a recursos externos também, como Google Drive.

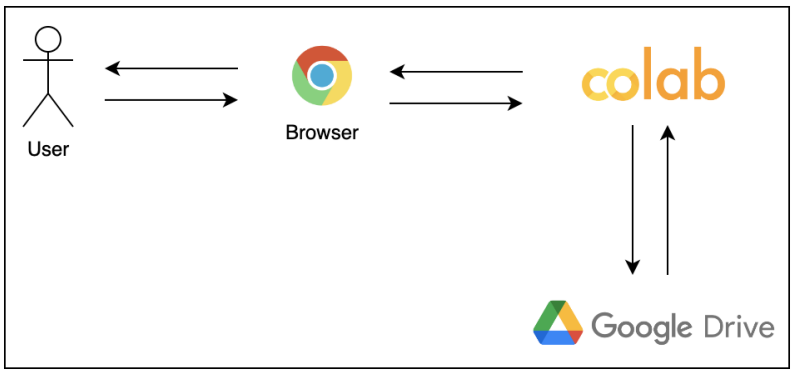

Figura 5: Arquitetura utilizando Google Colab

Podemos notar no esquema acima as seguintes características: a comunicação com o ambiente do Colab é efetuada por **HTTPS** (*HyperText Transfer Protocol Secure*), através de um browser. E por já estar no ecossistema Google, facilmente conseguimos integrar com ferramentas como Google Drive, que foi muito necessária para troca de arquivos no momento do treinamento do modelo.

E a segunda parte, que trata da arquitetura do software do Agente e seus módulos auxiliares, que estão em ambiente local, e interagem com outras ferramentas externas, como o Excel, ao gerar um arquivo de resultados finais, por exemplo.

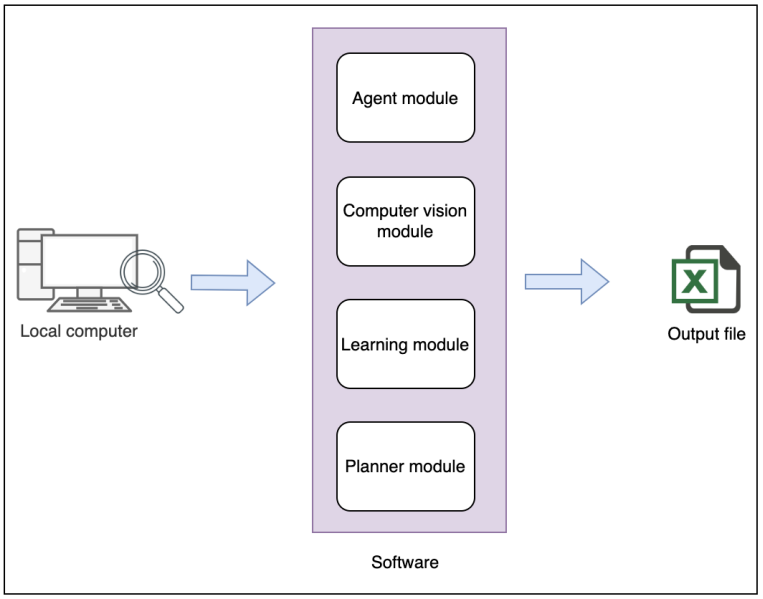

Figura 6: Arquitetura macro software local

4.1 Aspectos Funcionais e Não-Funcionais

Requisitos funcionais e não funcionais são dois tipos de requisitos que devem ser considerados na elaboração de um projeto de software.

Os requisitos funcionais são aqueles que descrevem as funcionalidades ou recursos que o sistema deve ter para atender às necessidades do usuário ou do negócio. Esses requisitos podem incluir funções específicas, como a capacidade de criar e editar dados, gerenciar contas de usuário ou fornecer relatórios de desempenho.
Já os requisitos não funcionais são aqueles que descrevem as qualidades ou características do sistema que não estão diretamente relacionadas às suas funcionalidades. Esses requisitos podem incluir questões como desempenho, segurança, confiabilidade, escalabilidade, usabilidade, entre outras. Os requisitos não funcionais não descrevem o que o sistema deve fazer, mas sim como ele deve fazê-lo.

Ambos os tipos de requisitos são igualmente importantes para o sucesso do projeto de software, e devem ser levados em consideração durante todas as fases do processo de desenvolvimento.

Ao analisar o sistema de maneira holística, temos algumas características que podemos classificar como funcionais ou não funcionais:

Funcionais	Não funcionais
Atuador do Agente	Engine do Angry Birds
Sensores do módulo de visão	Confiabilidade do modelo de visão
Máquina de estados	Desempenho do Agente

Tabela 1: Requisitos funcionais e não funcionais

4.2 Visão Estrutural

Algumas visões do software já foram citadas nos tópicos acima. Mas, neste capítulo será demonstrado graficamente as principais camadas envolvidas no desenvolvimento desta aplicação.

Podemos separar, inicialmente a estrutura geral do software, que contempla os seguintes módulos desenvolvidos:

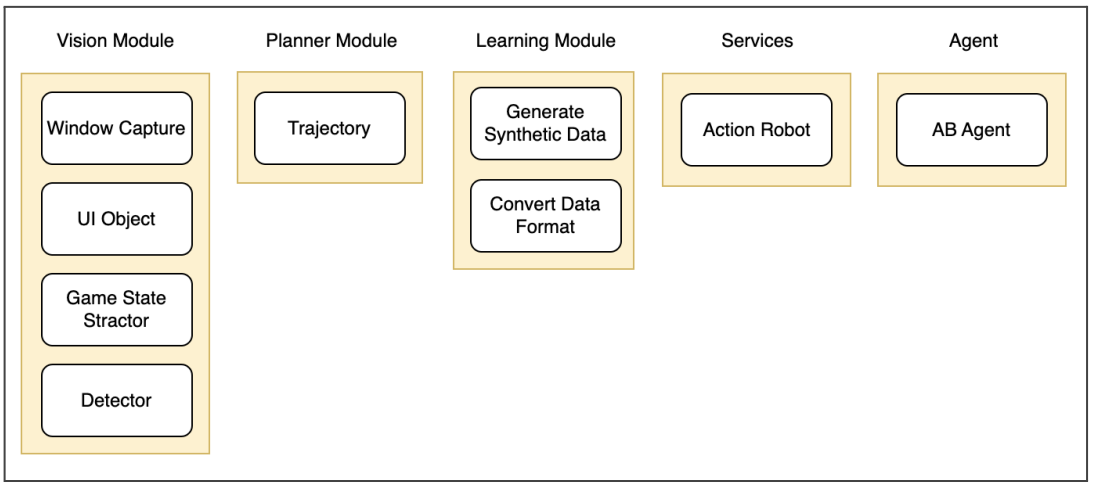

Figura 7: Visão estrutural geral

Em seguida, podemos demonstrar as entidades acessórias, que serviram de apoio para o desenvolvimento e para visualização de erros, bem como a correção de problemáticas:

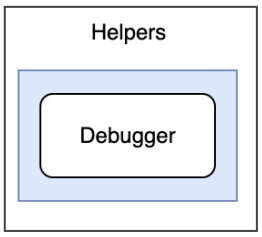

Figura 8: Visão estrutural das entidades acessórias

Por fim, temos as entidades externas, que nada mais são do que as principais bibliotecas utilizadas para identificar e executar ações na tela, bem como treinar o modelo de visão computacional:

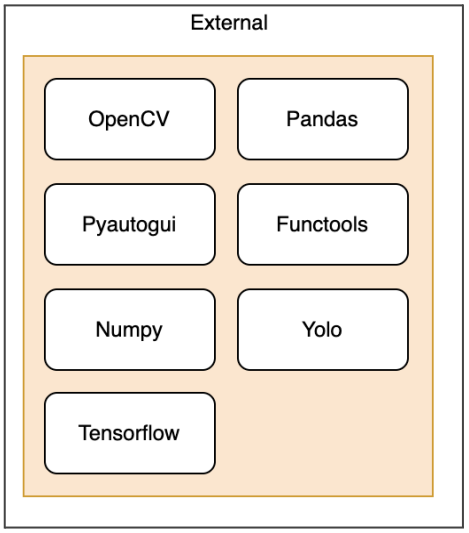

Figura 9: Visão estrutural externa

4.3 Visão Comportamental

Ao iniciar o jogo, o Agente interage com a tela de modo a selecionar o nível a ser jogado. Imediatamente após o carregamento da tela ele inicia a fase de observação, e com os dados coletados é capaz de definir a melhor estratégia para efetuar uma jogada. Idealmente os níveis deverão ser jogados seguinte os critérios:

- O Agente só pode jogar uma vez cada nível
- O Agente passa para o próximo nível quando concluir o anterior com sucesso

Perceba as etapas citadas no diagrama abaixo:

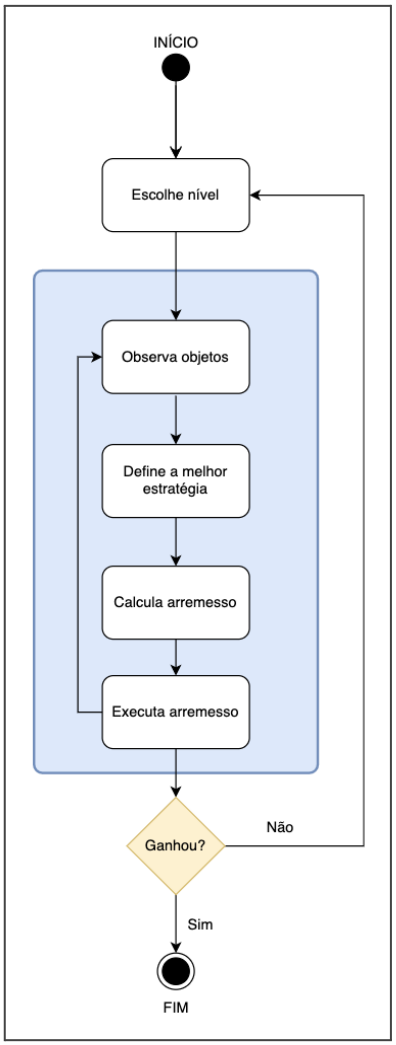

Figura 10: Comportamento do Agente

4.4 Visão de Implementação

Neste capítulo serão exploradas as ferramentas utilizadas para implementação deste projeto.

Podemos visualizar abaixo um esquema que representa as principais ferramentas e em qual momento do fluxo que elas foram utilizadas:

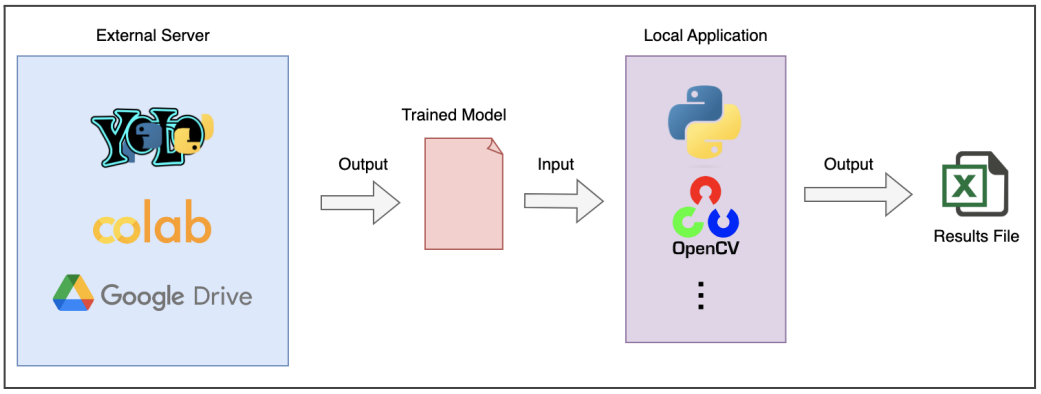

Figura 11: Visão de implementação do código

Como foi dito anteriormente o YOLO foi a ferramenta escolhida para o treinamento do modelo de visão computacional, e este treinamento se deu no ambiente do Google Colab.

Após o treinamento do modelo é gerado um arquivo que serve de input para a aplicação principal. Este software tem uma arquitetura monolítica, todavia foi desenvolvido em módulos, com baixo acoplamento entre eles e responsabilidades bem definidas, como podemos perceber em esquemas apresentados anteriormente.

Vamos explorar então alguns conceitos importantes para a concepção desta aplicação e a utilização de algumas bibliotecas.

4.5 DRY - Não se repita

O conceito **DRY** (*Don't Repeat Yourself*), em tecnologia, é uma abordagem que incentiva a evitar a repetição de informações ou lógicas de programação em um sistema [22]. Em outras palavras, significa que cada informação ou lógica deve ter uma única fonte de verdade, a fim de evitar redundâncias e aumentar a eficiência e a manutenção do código.

Ao seguir o princípio DRY, podemos reduzir o tempo e os recursos necessários para fazer alterações no sistema, pois não precisamos buscar e atualizar a mesma informação ou lógica em vários lugares. Isso também ajuda a minimizar erros e

inconsistências no código, já que qualquer mudança feita em uma fonte única será refletida em todo o sistema.

Em resumo, o princípio DRY é um conceito importante em tecnologia que incentiva a criação de sistemas eficientes, escaláveis e fáceis de manter, ao evitar a repetição desnecessária de informações ou lógicas de programação.

4.6 SOLID

O conceito de **SOLID** [23] é um conjunto de cinco princípios de programação orientada a objetos que visam a criação de sistemas mais flexíveis, fáceis de manter e testar. Cada letra do acrônimo SOLID representa um desses princípios:

- Princípio da responsabilidade única: cada classe deve ter uma única responsabilidade dentro do sistema, ou seja, deve ter apenas um motivo para mudar.
- Princípio do Aberto/Fechado: cada classe deve ser aberta para extensão, mas fechadas para modificação, isto seja, o comportamento da classe deve ser alterado por meio da adição de novas classes, não modificando as classes existentes.
- Princípio da Substituição de Liskov: as subclasses devem ser substituíveis por suas classes base sem afetar a integridade do sistema.
- Princípio da Segregação de Interface: as interfaces devem ser separadas em unidades lógicas menores, que contêm apenas os métodos necessários para a implementação da funcionalidade específica.
- Princípio da Inversão de Dependência: os módulos de alto nível não devem depender de módulos de baixo nível. Ambos devem depender de abstrações.

Ao seguir esses princípios, podemos criar sistemas mais flexíveis, escaláveis e fáceis de manter. Isso porque o código se torna mais modular, permitindo que as alterações sejam feitas com mais facilidade e sem afetar outras partes do sistema.

Além disso, isso ajuda a promover boas práticas de programação e a evitar problemas comuns, como acoplamento excessivo e dependência de implementações específicas.

5 Tecnologias e Aspectos de Implementação

Nesta seção vamos aprofundar um pouco mais nas tecnologias utilizadas, que ainda não foram citadas acima. Como Python, Numpy, OpenCV e TensorFlow, que foram muito importantes para o desenvolvimento do Agente, bem como seus sensores e atuadores.

5.1 Python

O **Python** é uma linguagem de programação de alto nível, interpretada e multiplataforma. É uma linguagem de programação bastante popular e amplamente utilizada em diversos campos, incluindo a inteligência artificial.

Esta linguagem oferece uma série de recursos e bibliotecas que são muito úteis para a programação de sistemas de inteligência artificial. Por exemplo, a biblioteca TensorFlow [24], uma das mais populares para aprendizado de máquina. Além disso, outras bibliotecas populares para aprendizado de máquina, como Keras e NumPy, também podem ser facilmente integradas em projetos Python para inteligência artificial.

Esta linguagem é fácil de aprender e de usar, e este foi um dos motivos da escolha de implementar neste projeto. Ademais, a sintaxe simples e legível do Python permitiu criar um código mais claro e organizado, facilitando a manutenção posterior.

5.2 NumPy

A biblioteca **NumPy** (Numerical Python) é uma das bibliotecas mais utilizadas na linguagem de programação Python para computação científica e análise de dados.

Ela fornece uma estrutura de dados multidimensional para manipulação de matrizes e vetores de forma eficiente e otimizada, permitindo a realização de operações matemáticas e científicas de maneira simples e rápida.

Algumas funcionalidades da biblioteca NumPy utilizadas neste sistema:

- **Array n dimensional**: uma estrutura de dados eficiente para representar matrizes e vetores.
- **Funções matemáticas para operações com arrays**: soma, subtração, multiplicação, divisão, entre outras.

5.3 OpenCV

O **OpenCV** (Open Source Computer Vision) é uma biblioteca de código aberto que fornece uma série de funções e algoritmos para processamento de imagens e visão computacional.

Desenvolvida originalmente pela Intel, esta biblioteca é amplamente utilizada em diversas áreas, incluindo robótica, automação industrial, vigilância, reconhecimento facial, entre outras.

Algumas funcionalidades da biblioteca OpenCV utilizadas neste sistema:

- Carregamento e exibição de imagens e vídeos.
- Pré-processamento de imagens: transformação de cores, filtragem, redimensionamento, recorte, entre outros.
- Detecção de bordas e contornos.
- Segmentação de objetos em uma imagem.
- Rastreamento de objetos em movimento.

5.4 TensorFlow

O **TensorFlow** é uma biblioteca de software de código aberto para aprendizado de máquina e inteligência artificial. Ela foi desenvolvida pela equipe do Google Brain [25] e lançada em 2015. Esta biblioteca é amplamente utilizada em diversas áreas, incluindo processamento de linguagem natural, visão computacional, reconhecimento de fala, entre outras.

Sua principal característica é a sua capacidade de realizar cálculos em grandes volumes de dados, especialmente em redes neurais profundas, o que permite treinar modelos complexos em grande escala.

O TensorFlow possui uma API (*Application Programming Interface*), que nos permite desenvolver modelos de aprendizado de máquina e inteligência artificial usando uma variedade de algoritmos, incluindo redes neurais convolucionais, redes neurais recorrentes, modelos de sequência a sequência, entre outros. E isto é utilizado no YOLO, por exemplo.

Sendo assim, a relação entre o TensorFlow e o YOLO é a seguinte: o TensorFlow cria e treina modelos de detecção de objetos. O YOLO, por sua vez, é o algoritmo utilizado, baseado em redes neurais convolucionais.

6 Testes e Análise dos Resultados

Nesta seção serão detalhados os testes realizados, bem como a análise dos resultados encontrados para cada estratégia definida.

Foi definido que o sistema de pontuação do jogo, bem como a validação se a estratégia do Agente ganhou ou não determinado nível servirá de base para a análise do Agente em si.

A primeira estratégia foi a que sofreu maior variação em quesito de pontuação e finalização do jogo, por não ter uma previsibilidade de jogada. Foi observado que,

muitas vezes, a trajetória escolhida ao acaso não ia em direção ao alvo, prejudicando assim a performance geral do Agente.

O demonstrativo disso é a tabela abaixo:

Nível	Pontuação máxima	Ganhou
1	27850	Sim
2	30149	Sim
3	39067	Sim
4	40003	Sim
5	**43550**	Não
6	24501	Não
7	37348	Não
8	30298	Não
9	23670	Não
10	38651	Não

Tabela 2: Análise de resultados da estratégia 1

A segunda estratégia continha um elemento mais previsível, que era o limite de ângulos. Sem poder jogar para cima, por exemplo, ou até mesmo para o lado oposto ao alvo, podemos perceber que a pontuação mínima do Agente cresceu consideravelmente:

Nível	Pontuação máxima	Ganhou
1	28765	Sim
2	45123	Sim
3	46184	Sim
4	46348	Sim
5	46538	Não

6	**54689**	Sim
7	54276	Sim
8	52765	Não
9	49877	Não
10	48965	Não

Tabela 3: Análise de resultados da estratégia 2

A última estratégia contou com a implementação do elemento de visão computacional. É possível perceber que neste modelo o Agente conseguiu executar todos os níveis com pontuações expressivamente mais altas.

Isto se deve a um conjunto de coisas nesta estratégia:

- Considerar jogadas anteriores.
- Considerar o módulo de visão computacional com um nível de confiança alto (entre 80% e 85%).

Nível	Pontuação máxima	Ganhou
1	31298	Sim
2	46234	Sim
3	47654	Sim
4	48009	Sim
5	50287	Sim
6	56762	Sim
7	57134	Sim
8	58156	Sim
9	**58308**	Sim
10	56721	Sim

Tabela 4: Análise de resultados da estratégia 3

7 Conclusões

Ao realizar este trabalho foi possível chegar às seguintes conclusões:

- Quanto maior o conjunto de dados que utilizamos no treinamento do modelo computacional, mais fácil ficava do módulo de visão encontrar os elementos que mapeamos.
- Quanto mais elaborada a estratégia do Agente, mais rápido a conclusão de um nível com sucesso, mas não necessariamente com maior pontuação. Afinal, não existe estratégia perfeita, capaz de mapear todos os casos previamente.

Além disso, ao modificar o nível de confiança do módulo de visão computacional o Agente obteve resultados diferentes. Nem sempre o maior nível de confiança era o ideal para acertar o alvo todas as vezes.

O melhor comportamento foi identificado com o nível de confiança entre 80% e 85%, para um conjunto de dados de 5.000 itens, visto que, aumentando demais este nível de confiança o Agente não conseguia identificar objeto algum, e abaixando demais o Agente identifica incorretamente os objetos classificados anteriormente (porcos).

Esta é uma característica muito comum quando se trabalha com detecção de objetos. Não existe um número comum que resolva todos os problemas, de modo genérico. Foi preciso diversos testes e ajustes para chegar a conclusão do que seria ideal para o presente trabalho. Ao diminuir o conjunto de dados, por exemplo, foi possível observar o mesmo comportamento descrito acima, o que confirma a necessidade de ajustes a cada etapa.

As dificuldades encontradas estavam relacionadas ao tempo, hardware e ambientes disponíveis, o que nos leva a pensar que em situações ideais, os resultados seriam ainda melhores. Além disso, a arquitetura foi construída de tal modo, em que a estratégia do Agente poderia ser facilmente substituída, deixando os módulos de visão e ação na tela reutilizáveis.

Isto abre caminho para novos testes, e pode contribuir para comunidade de modo que utilizem o código gerado para explorar outros aspectos do jogo, como performance, ganhar o jogo com menos jogadas possíveis, entre outras coisas.

Referências Bibliográficas

[1] RUSSELL, Stuart J.; NORVIG, Peter. Artificial intelligence: a modern approach. Prentice- Hall, 3rd Edition, 2009.

[2] Angry Birds. Disponível em: <https://www.angrybirds.com/>. Acesso em: 31 mar. 2023.

[3] IRIE, Kiyoshi. “Yolo V4, v3 and v2 for Windows and Linux.” *GitHub*, 24 Jan. 2023, github.com/kiyoshiiriemon/yolov4_darknet. Accessed 3 Jan. 2023.

[4] GOOGLE. Google Colaboratory. Disponível em: <https://colab.research.google.com/>.

[5] BICALHO, L. F.; FEIJÓ, B.; BAFFA, A. A Culture Model for Non-Player Characters’ Behaviors in Role-Playing Games. Disponível em: <https://ieeexplore.ieee.org/document/9291553/>. Acesso em: 31 mar. 2023.

[6] Xia, Boming, Xiaozhen Ye, and Adnan OM Abuassba. "Recent research on ai in games." 2020 International Wireless Communications and Mobile Computing (IWCMC) (2020): 505-510.

[7] YANG, Q. Transfer learning. [s.l.] Cambridge Cambridge University Press, 2020.

[8] ELGENDY, M. Deep learning for vision systems. [s.l.] Shelter Island, Ny Manning Publications Co, 2020.

[9] STERNBERG, R. J.; ROBERTO CATALDO COSTA; VITOR GERALDI HAASE. Psicologia cognitiva. [s.l.] Porto Alegre, Artmed, 2008.

[10] KORKUT, E. H.; SURER, E. Visualization in virtual reality: a systematic review. Virtual Reality, 17 jan. 2023.

[11] BHARATI, P.; PRAMANIK, A. Deep Learning Techniques—R-CNN to Mask R-CNN: A Survey. Computational Intelligence in Pattern Recognition, p. 657–668, 2019.

[12] COCO - Common Objects in Context. Disponível em: <https://cocodataset.org>.

[13] NAUGHTY DOG. Naughty Dog. Disponível em: <https://www.naughtydog.com/>. Acesso em: 31 mar. 2023.

[14] BARBOSA, S. T.; VEIGA, J.; CARVALHO, C. V. D. A. Estudo do Uso de Técnicas de Inteligência Artificial em Jogos 2D. Revista Eletrônica TECCEN, v. 5, n. 1, p. 05, 1 abr. 2012.

[15] FELIPE CANDIDO, J. Desenvolvimento de Agente Inteligente Capaz de Jogar Angry Birds.

[16] MENEZES, Richardson Santiago Teles de. ChessPy: ferramenta para detecção inteligente de peças de xadrez. 2022. Trabalho de Conclusão de Curso. Universidade Federal do Rio Grande do Norte.

[17] Google Research. Disponível em: <https://research.google/>.

[18] PYTHON. Welcome to Python.org. Disponível em: <https://www.python.org/>.

[19] NUMPY. NumPy. Disponível em: <https://numpy.org/>.

[20] OPENCV. OpenCV library. Disponível em: <https://opencv.org/>.

[21] ALVES, G. Detecção de Objetos com YOLO - Uma abordagem moderna. Disponível em: <https://iaexpert.academy/2020/10/13/deteccao-de-objetos-com-yolo-uma-abordagem-moderna/?doing_wp_cron=1692750706.7121419906616210937500>. Acesso em: 31 mar. 2023.

[22] DRY. Disponível em: <https://thevaluable.dev/dry-principle-cost-benefit-example/>. Acesso em: 31 mar. 2023.

[23] MARTIN, R. C. Clean code a handbook of agile software craftmanship. [s.l.] Upper Saddle River [Etc.] Prentice Hall, 2010.

[24] TensorFlow. Disponível em: <https://www.tensorflow.org/?gclid=Cj0KCQjwiZqhBhCJARIsACHHEH8rRtlNSA_vew71kEw-L1C8RtV-lsdeemx4nal9c-neV-ufqG28WRMaAm3yEALw_wcB&hl=pt-br>. Acesso em: 31 mar. 2023.

[25] Brain Team – Google Research. Disponível em: <https://research.google/teams/brain/>. Acesso em: 31 mar. 2023.

[26] ARTASANCHEZ, A.; JOSHI, P. Artificial intelligence with Python : your complete guide to building intelligent apps using Python 3.x and TensorFlow 2. Birmingham: Packt Publishing, 2020.

[27] KARLSSON, Börje Felipe Fernandes. Um middleware de inteligência artificial para jogos digitais. 2005. Tese de Doutorado. Masters thesis. Informatics Department.

[28] REIS, Mariana De Luca; DE OLIVEIRA ANDRADE, Kleber. Áreas de Pesquisa e técnicas de inteligência artificial em jogos digitais. Revista Tecnológica da Fatec Americana, v. 10, n. 01, p. 71-97, 2022.

[29] MNIH, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., & Riedmiller, M. (2013). Playing atari with deep reinforcement learning.arXiv preprint arXiv:1312.5602.

[30] LEE, Joo-Seon; SEON, Hyun-Su; KIM, Jeong-Hyeon; JOO, Su-Young; KIM, Kyung-Joong. Angry Plan A+ Team – Team Description Paper. Symposium on AI in Angry Birds, 2014. 1 p.

[31] XiaoYu Ge, Stephen Gould, Jochen Renz, Sahan Abeyasinghe, Jim Keys, Andrew Wang, Peng Zhang, Angry Birds Game Playing Software Version 1.32, aibirds.org, 2014.

[32] BOROVIČKA, Tomáš; ŠPETLÍK, Radim; RYMEŠ, Karel. DataLab Birds Angry Birds AI. Symposium on AI in Angry Birds, 2014. 2 p.