

UNIVERSIDADE FEDERAL DO PAMPA
Ricardo de Oliveira Dora

Uma biblioteca para a criação de jogadores virtuais em emuladores

Alegrete
2018

Ricardo de Oliveira Dora

Uma biblioteca para a criação de jogadores virtuais em emuladores

Trabalho de Conclusão de Curso apresentado ao Curso de Graduação em Ciência da Computação da Universidade Federal do Pampa, como requisito parcial para obtenção do Título Bacharel em Ciência da Computação.

Orientador: Prof. Dr. Marcelo Resende Thielo

Alegrete

2018

Ricardo de Oliveira Dora

Uma biblioteca para a criação de jogadores virtuais em emuladores

Trabalho de Conclusão de Curso apresentado ao Curso de Graduação em Ciência da Computação da Universidade Federal do Pampa, como requisito parcial para obtenção do Título Bacharel em Ciência da Computação.

Projeto de Trabalho de Conclusão de Curso defendido e aprovado em dede

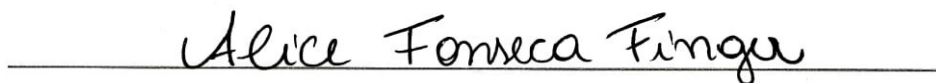
Banca examinadora:



Prof. Dr. Marcelo Resende Thielo

Orientador

UNIPAMPA



Prof.ª Ma. Alice Fonseca Finger

UNIPAMPA



Prof. Me. Jean Felipe Patikowski Cheiran

UNIPAMPA

Agradecimentos

Agradeço aos meus pais, à minha irmã, à minha companheira, ao professor Dr. Marcelo Resende Thielo, à Luci do Nude e à minha psicóloga Cris pelo apoio.

“As grandes idéias surgem da observação dos pequenos detalhes”.

(Augusto Cury)

RESUMO

Dentre as áreas estudadas na Ciência da Computação, a Inteligência Artificial tem recentemente se destacado não apenas na comunidade acadêmica como também na indústria, em especial na área de jogos. Tendo em vista que há demanda para ambientes de inteligência artificial para criação e testes de novas tecnologias, o presente trabalho de conclusão de curso tem como objetivo a definição e construção de uma biblioteca de software que servirá como interface para a criação e testes de algoritmos de inteligência artificial com o emulador *Multiple Machine Arcade Emulator* (MAME). Foi escolhido por ser o emulador com mais jogos disponíveis e maior variedade de arquiteturas de hardware, pois emula diversas plataformas diferentes, onde cada jogo poderá servir como um novo ambiente, com suas próprias regras. A fim de atingir este objetivo, foi desenvolvida uma biblioteca em Python capaz de interagir com os jogos do MAME, entregando a possibilidade de utilizar os quadros do jogo e enviar eventos de teclado para o emulador, para interagir com o jogo que está sendo executado. Este trabalho contribui com a definição e implementação de uma biblioteca Python que pode ser usada por qualquer um que queira construir algoritmos de inteligência artificial para jogos do emulador MAME.

Palavras-Chave: Inteligência Artificial. Emulação. MAME. Fliperama.

ABSTRACT

Among the areas studied in Computer Science, Artificial Intelligence has recently stood out not only in the academic community but also in industry, especially in the area of games. Considering that there is demand for artificial intelligence environments to create and test new technologies, the present bachelor thesis has the purpose of defining and building a software library that will serve as an interface for the creation and testing of algorithms of artificial intelligence with the Multiple Machine Arcade Emulator (MAME), which is the emulator with more games available and greater variety of hardware architectures, since it emulates several different platforms, in which each game can serve as a new environment, with its own rules. In order to achieve this goal, a Python library has been developed that can interact with MAME games, offering the possibility to use the game frames and send keyboard events to the emulator to interact with the game being played. This work contributes with the definition and implementation of a Python library that can be used by anyone who wants to built artificial intelligence algorithms for MAME emulator games.

Keywords: Artificial intelligence. Emulation. MAME. Arcade.

LISTA DE FIGURAS

Figura 1 - Máquina de Fliperama	14
Figura 2 - Menu Principal do MAME	18
Figura 3 - Descrição de nível em VGDL	24
Figura 4 - Arquitetura do Pogamut 3	26
Figura 5 - Mapeamento de botões de um jogo de arcade	31
Figura 6 - Funções Principais	32
Figura 7 - Captura de Tela	36
Figura 8 – Templates para o jogo Super Tank	37
Figura 9 - Detecção de Objetos	38
Figura 10 - Detecção de Objetos com Cor	39
Figura 11 - <i>Template</i> para função de detecção com transparência	40
Figura 12 - Detecção de Objetos com Transparência	41

LISTA DE TABELAS

Tabela 1 - Cronograma de Atividades	28
---	----

LISTA DE SIGLAS

AI Artificial Intelligence
ALE Arcade Learning Environment
BIOS Basic Input/Output System
CHD Compressed Hunks of Data
DLL Dynamic Link Library
FICS Free Internet Chess Server
FIDE Federação Internacional de Xadrez
GVGAI General Video Game AI
JMX Java Management Extensions
IA Inteligência Artificial
MAME Multiple Arcade Machine Emulator
MESS Multi Emulator Super System
RLE Retro Learning Environment
ROM Read-only memory
VGDL Video Game Definition Language

Sumário

1. Introdução.....	12
1.1 Motivação.....	13
1.2 Objetivo Geral.....	15
1.3 Objetivos Específicos.....	15
1.4 Organização do Documento.....	16
2. Fundamentação Teórica.....	17
2.1 Emulação.....	17
2.2 MAME.....	18
2.2.1 MESS.....	18
2.2.2 ROMs.....	19
2.2.3 BIOS.....	19
2.2.4 Questões Legais.....	20
2.2.5 Integração com Lua.....	21
2.3 Bibliotecas.....	21
3. Trabalhos Relacionados.....	23
4. Metodologia.....	27
4.1 Software.....	28
4.2 Hardware.....	29
5. Desenvolvimento do Trabalho.....	30
5.1 Definição das ferramentas utilizadas.....	30
5.2 Arquitetura da Biblioteca.....	32
5.2.1 Módulo de simulação de eventos de teclado.....	33
5.2.2 Módulo de mapeamento de teclas.....	34
5.2.3 Módulo de captura de tela.....	34
5.3 Apresentação e Testes das Funções.....	35
5.3.1 Função de Simulação de Eventos de Teclado.....	35
5.3.2 Função de Captura de Tela.....	35
5.3.3 Função de Detecção de Objetos.....	36
5.3.4 Função de Detecção de Objetos com Cor.....	38
5.3.5 Função de Detecção de Objetos com Transparência.....	39

6. Resultados.....	42
6.1 Algoritmo.....	42
6.2 Execução na Resolução Padrão do MAME.....	42
6.3 Execução na Resolução Nativa do Jogo.....	43
7. Considerações Finais.....	44
Referências.....	46
Apêndices.....	49
APÊNDICE A – Instruções para instalar a biblioteca.....	49
APÊNDICE B – Instruções para utilizar a biblioteca.....	51

1. Introdução

Nos últimos anos, temos observado uma explosão de usos de algoritmos de aprendizagem de máquina e inteligência artificial (IA) em diversas áreas, com numerosas e distintas aplicações. Segundo Gomes (2010):

A Inteligência Artificial é uma das ciências mais recentes, tendo início após a Segunda Guerra Mundial e, atualmente, abrangendo uma enorme variedade de subcampos, desde áreas de uso geral, como aprendizado e percepção, até tarefas específicas como jogos de xadrez, demonstração de teoremas matemáticos, criação de poesia e diagnóstico de doenças.

A crescente demanda por profissionais dessa área, o grande investimento de gigantes da tecnologia como *Facebook*, *Google* e *IBM* e o fato de ser uma das áreas mais intrigantes da pesquisa em ciência da computação são fatores que justificam os esforços para o desenvolvimento de avanços na área. Segundo o artigo publicado no *The New York Times* por Metz (2017), há uma escassez de talentos na área de inteligência artificial, e as grandes empresas estão dispostas a pagar milhões de dólares por especialistas neste campo de atuação. No Brasil não é diferente, de acordo com a Agência O Globo (2017), em matéria publicada no final de 2017, a previsão era de que em 2018 o volume de recursos destinados ao setor de inteligência artificial iria aumentar em mais de cinco vezes em relação ao ano da publicação.

Controlar dados de entrada não tratados (do inglês *raw input data*), como imagem ou som, é uma tarefa importante para a inteligência artificial, havendo a possibilidade de utilização em aplicações do mundo real, como a condução autônoma de veículos e navegação. No entanto, o desenvolvimento de agentes capazes de interagir com o mundo real muitas vezes pode ser caro ou inviável (Bhonker; Rozenberg; Hubara, 2016). Assim sendo, a interação simulada por um ambiente virtual pode ser fundamental para o desenvolvimento de novas tecnologias.

Este trabalho apresenta como proposta a criação de uma componente de software (e.g. biblioteca) para um emulador de máquinas de vídeo jogos de fliperama com o intuito de adicionar funcionalidades que poderão auxiliar no desenvolvimento e avaliação de algoritmos dos mais diversos tipos, com ênfase em Inteligência Artificial. A escolha de utilizar o emulador MAME para a criação do projeto é determinada pelo fato do mesmo emular muitas arquiteturas diferentes e

pelo grande número de jogos disponibilizados, atualmente mais de 5000, cada um sendo capaz de servir como um novo ambiente virtual.

1.1 Motivação

A relação entre jogos e inteligência artificial não é recente. Shannon (1950) já acreditava que o xadrez era o experimento ideal para testar algoritmos de Inteligência Artificial por possuir algumas características, como: 1) regras fortemente definidas, tanto nas operações permitidas quanto no objetivo final (xeque-mate); 2) não ser tão simples a ponto de ter uma solução trivial ou complexo demais para detectar uma solução satisfatória. Essa relação teve reflexo inclusive na cultura popular, como por exemplo no filme *Jogos de Guerra* (Badham, 1983), onde o personagem principal, pensando estar conectado em uma empresa de jogos, acaba inadvertidamente invadindo um computador do sistema de defesa norte-americano que utilizava inteligência artificial para planejar ataques e contra-ataques nucleares, escolhendo jogar “Guerra Global Termonuclear” ao invés do xadrez (sugerido pelo computador), e se envolve em uma simulação de guerra termonuclear que pode ter um fim trágico.

Outro motivo que faz o xadrez ser tão importante para a IA é que possui um método bem desenvolvido para classificar a força dos jogadores, chamado *Elo rating* (ENSMENGER, 2012), o qual é a base de sistemas utilizados pela Federação Internacional de Xadrez (FIDE) e *Free Internet Chess Server* (FICS), por exemplo. Esse método disponibilizou uma forma numérica clara e bem aceita pela comunidade para avaliar o progresso dos programas que jogavam xadrez.

Essas características que fizeram o xadrez ser tão importante para o estudo da IA são compartilhadas com diversos jogos de fliperama. Um exemplo de máquina de fliperama pode ser visto na Figura 1. Entre as principais características compartilhadas pelo xadrez e por jogos de fliperama destacam-se: as operações bem definidas, que seriam os comandos do jogador; objetivo final bem definido que varia de jogo para jogo, podendo ser algo como passar de fase ou atingir o maior número de inimigos na tela e, principalmente, uma forma numérica para avaliar o progresso, que seria a pontuação do jogo, conhecida como *score*. O xadrez, apesar de ser um excelente jogo para *benchmarking* de IA, possui limitações que restringem os seus casos de aplicação. Outros jogos, mais recentes, possuem como suas características maior dinamicidade, e também uma maior quantidade de

possibilidades de resultados por jogada. No entanto, para que se possa utilizar jogos eletrônicos mais recentes com algoritmos de IA, é necessário que seja construída uma forma de interfaceamento, através de componentes de software, como por exemplo uma biblioteca.

Figura 1 – Máquina de Fliperama



Fonte: (Site MAMEDEV, 2018)¹

A criação do software proposto neste trabalho proporcionará aos seus usuários um ambiente de desenvolvimento integrado com o emulador de máquinas de fliperama, facilitando a visualização dos resultados de seus algoritmos para diversos jogos, e fornecendo um laboratório virtual para *benchmarking* de algoritmos e técnicas de aprendizagem de máquina, inteligência artificial e outras áreas.

Para estimular pesquisas na área da inteligência artificial foram criadas diversas competições, dentre as quais se encontram competições específicas de desempenho de algoritmos de IA em jogos como, por exemplo, a *General Video Game AI* (GVGAI) (Perez-Liebana; Samothrakis; Togelius; Lucas; Schaul, 2016) em que o propósito é a criação de uma inteligência artificial capaz de jogar qualquer jogo, e a *BotPrize* (Hingston, 2009), que pode ser considerada uma variante do Teste de Turing para jogos tiro em primeira pessoa, pois foca na criação de um algoritmo que seja capaz de exibir um comportamento inteligente equivalente ao comportamento de um humano, a ponto de convencer um painel de juízes que é

¹ Disponível em <<https://www.mamedev.org/roms/supertnk/>>

realmente um jogador humano. Uma das contribuições da ferramenta proposta poderá estar relacionada a tais competições, aumentando significativamente a gama de jogos que poderão servir de base para a criação e testes dos algoritmos.

Além disso, professores poderão utilizar essa ferramenta em propostas de trabalhos, para que os seus alunos aperfeiçoem a compreensão dos fundamentos de suas disciplinas. Também servirá como uma ferramenta de apoio para o desenvolvimento de novos algoritmos de IA para novos jogos e para a pesquisas na área. Algumas subáreas de pesquisa, como o aprendizado de máquina, visão computacional, planejamento automatizado e busca (*pathfinding*) também poderão se beneficiar com a ferramenta proposta.

1.2 Objetivo Geral

Esta pesquisa visa a definição e construção de uma biblioteca que servirá como interface para possibilitar aos alunos e pesquisadores da área de IA a realização de experimentos relacionados a seus estudos.

A biblioteca deve permitir a interação com uma grande diversidade de jogos de fliperama, permitindo interagir com os jogos em tempo de execução, de forma fácil, intuitiva e mais abrangente, em termos quantidade de jogos, do que os métodos do estado da arte.

1.3 Objetivos Específicos

- Identificar as melhores tecnologias para a criação da biblioteca (linguagem de programação, emulador)
- Definir a linguagem na qual a biblioteca aceitará os Scripts
- Identificar a melhor maneira para ler os dados da memória do emulador
- Identificar a melhor maneira de fazer com que a biblioteca atue (fazer com que o emulador reconheça os comandos externos)
- Identificar técnicas para obter os *frames* do jogo de maneira eficiente
- Construir um protótipo para testar as funcionalidades da biblioteca em jogos distribuídos livremente.

1.4 Organização do Documento

Capítulo 2 (Fundamentação Teórica): Neste capítulo são explicados os conceitos teóricos que serão utilizados no trabalho, como emuladores, MAME e bibliotecas.

Capítulo 3 (Trabalhos Relacionados): Neste capítulo são apresentados os trabalhos relacionados aos temas de pesquisa do trabalho, destacando suas contribuições.

Capítulo 4 (Metodologia): Neste capítulo são apresentadas as atividades e ferramentas que serão consideradas para o desenvolvimento do trabalho.

Capítulo 5 (Desenvolvimento do Trabalho): Neste capítulo são apresentados os detalhes do desenvolvimento da biblioteca, sua arquitetura e funções.

Capítulo 6 (Resultados): Neste capítulo são apresentados os resultados obtidos após o término dos testes com a biblioteca finalizada.

Capítulo 7 (Considerações Finais): Neste capítulo são apresentadas as considerações finais sobre a contribuição do trabalho.

2. Fundamentação Teórica

Neste capítulo serão apresentados os conceitos teóricos utilizados no decorrer do trabalho, a fim de contextualizar o leitor. Compreender esses conceitos será fundamental para o entendimento geral do presente trabalho.

2.1 Emulação

De acordo com Baer (2007), emulação é o processo de implementar a interface e a funcionalidade de um sistema ou subsistema em um sistema ou subsistema que possui uma interface e funcionalidade diferentes. Segundo o site da biblioteca nacional dos Países Baixos, (Koninklijke Bibliotheek, 2017), a emulação é melhor descrita como a tentativa de imitar uma determinada plataforma ou programa de computador em outra plataforma ou programa, e é apresentado como uma das soluções para o problema da preservação digital, que se tem como finalidade “assegurar proteção à informação de valor permanente para acesso pelas gerações presentes e futuras” (Hedstrom, 1997).

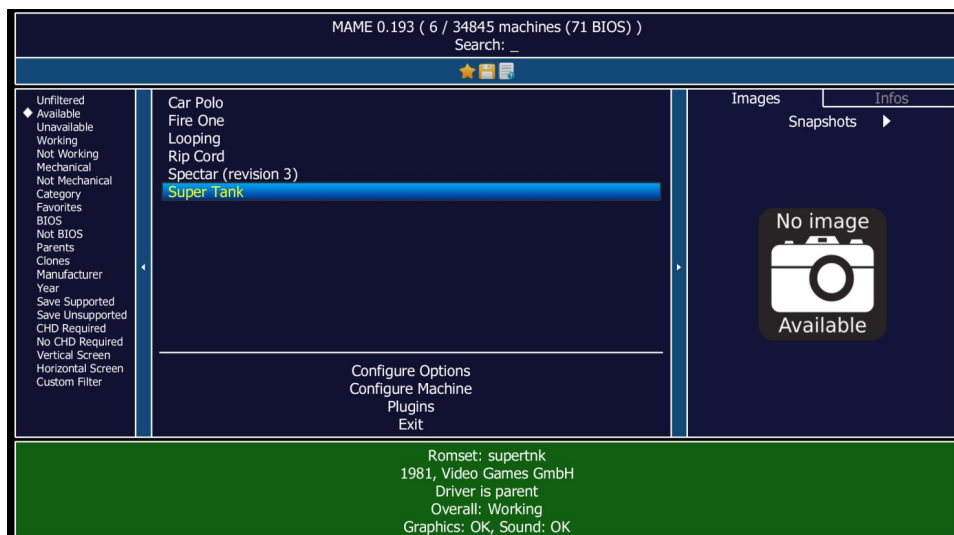
Os softwares frequentemente são criados para funcionar em hardwares específicos. Os emuladores foram criados para possibilitar que softwares feitos para hardwares específicos possam funcionar em outros ambientes. O primeiro emulador foi criado em 1964, pela IBM, para fazer com que os programas criados para as máquinas IBM 7070 funcionassem nos seus novos mainframes IBM System/360.

Os emuladores focam em recriar o *design* interno do sistema (hardware, software ou ambos) ou em criar um ambiente que possibilite que o *software* original possa ser executado, com seu código original, como se estivesse sendo executado na arquitetura original, ao contrário do que faz um simulador, que pode recriar todo o programa para que se comporte como o original, de forma fiel, mas feito em um outro ambiente. Por exemplo, um jogo como *Pacman* de fliperama poderia ser reescrito para um computador e proporcionar a mesma experiência do jogo original, sendo assim uma simulação. A emulação, por outro lado, deve disponibilizar um ambiente adequado para executar o software original do jogo *Pacman*, sem modificações (Murphy, 2013).

2.2 MAME

Multiple Machine Arcade Emulator (MAME) é um emulador de código aberto projetado originalmente para recriar o comportamento do hardware de sistemas de jogos de arcade em software para uso em computadores pessoais. Seu principal objetivo é servir de referência ao funcionamento interno das máquinas emuladas, tanto para fins educacionais quanto para fins de preservação, para evitar que décadas de história em sistemas de entretenimento sejam perdidos, uma vez que o hardware em que são executados eventualmente irá parar de funcionar. O código fonte do *MAME* é considerado como sendo a documentação do hardware e de como ele funciona, e o fato do software ser utilizável é considerado como um bom efeito colateral, mas não o foco principal da existência do *MAME*, apesar de poder ser utilizado para validar a precisão da documentação (MAMEdev Team, 2018). Na Figura 2 é possível ver o menu principal do *MAME*.

Figura 2 – Menu Principal do *MAME*



Fonte: Elaborado pelo autor.

2.2.1 MESS

Multi Emulator Super System (MESS) é um emulador que documenta e reproduz componentes internos de diversos sistemas computacionais, como computadores, consoles e até mesmo calculadoras. De forma semelhante ao *MAME*, o *MESS* permite usar em computadores pessoais modernos os programas e jogos que foram desenvolvidos para diversas máquinas, sendo atualmente capaz de

emular mais de 550 sistemas das últimas 5 décadas. (MESS, 2017)

Ao longo do tempo, o *MESS* foi descontinuado como projeto independente e encontra-se formalmente integrado ao *MAME*, tornando-se parte do mesmo em maio de 2017.

2.2.2 ROMs

Read-only memory (ROM) significa memória somente de leitura. No contexto da emulação, são arquivos de imagem contendo os jogos ou os softwares que serão executados nos emuladores e levam esse nome pois os *chips* usados para armazenar os dados dos jogos não eram regraváveis pelo usuário e eram permanentes. (MAMEdev, 2018)

Imagem *ROM*, também chamado de arquivo *ROM*, é a imagem extraída de circuitos integrados tipo *ROM*, *PROM*, *EPROM* ou *EEPROM* contidos em diversos dispositivos eletrônicos como, por exemplo, cartuchos de *videogames*. O procedimento para a extração dos dados dos jogos é chamado de *dumping* e, para a maioria dos sistemas de videogame domésticos comuns, é possível realizar o *dumping* através de gravadores de *EPROM* ou dispositivos como o *Doctor V64*, que é capaz de transformar o conteúdo dos *chips* de cartuchos do console *Nintendo 64* em uma série de instruções codificadas que podem ser executadas posteriormente em um hardware diferente ou em um emulador (Fassone, 2015).

Em se tratando de jogos de *arcade*, a imagem *ROM* contém uma cópia de todos os dados de dentro de um determinado *chip* ou placa-mãe de *arcade*. Alguns jogos utilizam mídias de armazenamento adicionais e, por motivos técnicos, essas mídias são inadequadas para serem armazenadas da mesma forma que os dados da *ROM*. Por essa razão, um novo formato foi criado, chamado de *Compressed Hunks of Data* (*CHD*), que serve para guardar os dados de áudio e vídeo, como trilhas sonoras ou sequências de vídeo não interativas dentro dos jogos, chamadas de *cinematics* ou *cutscenes* (MAMEdev, 2018).

2.2.3 BIOS

Segundo Laureano (2006), em uma máquina real, o Sistema Básico de Entrada e Saída, do inglês, *Basic Input/Output System* (*BIOS*), exerce a função de fornecer as operações de baixo nível para que um sistema operacional possa

acessar os vários recursos da placa-mãe, memória e serviços de entrada e saída. Consiste em um *chip* de memória que mantém seus dados quando a energia é desligada, ou seja, não-volátil. Originalmente a *BIOS* era armazenada em *chips* de memória somente de leitura, mas em computadores novos o seu conteúdo é armazenado em memória *flash*, para possibilitar a reescrita sem a necessidade de remover o *chip* da placa-mãe.

Para diversos emuladores de videogame e para a maioria dos sistemas emulados pelo *MAME*, é necessário conseguir a *BIOS* do sistema original, através de engenharia reversa ou do *dump* do *chip*, de forma semelhante à *ROM* (*MAMEdev*, 2018).

2.2.4 Questões Legais

De acordo com o informe técnico (Conley et al., 2004), o argumento de que a emulação preserva os jogos de videogame, muitos dos quais, de outra forma, estariam se aproximando da extinção, é relativamente fraco, já que somente os detentores dos direitos autorais podem determinar se desejam que seu software seja arquivado. Também é mencionado que a emulação é diferente da pirataria de software, pois não é inerentemente ilegal. A combinação específica e o uso de seus componentes individuais determinam a legalidade de um emulador, interpretando assim que o emulador por si só é permitido, contanto que não seja incorporado no próprio emulador códigos do console original, como a *BIOS* e o uso de jogos protegidos por direitos autorais através das *ROMs*.

Por esse motivo o *MAME* não fornece nenhum software para ser executado nas máquinas emuladas. Porém, há opções para obter legalmente jogos para utilizar no *MAME*, sendo possível baixar *ROMs* liberadas gratuitamente para o público para fins não comerciais. No site oficial do *MAME* encontram-se atualmente 20 jogos disponíveis nessas condições, e também há a possibilidade de obter uma licença para os jogos comprando através de um distribuidor ou fornecedor. Algumas coleções de jogos clássicos que podem ser compradas legalmente em serviços como *Steam* e *Humble Bundle* são compatíveis com o *MAME* e outros emuladores. (*MAMEdev*, 2018).

2.2.5 Integração com Lua

Lua é uma linguagem de programação criada em 1993 no Brasil, na Pontifícia Universidade Católica do Rio de Janeiro (PUC-Rio). Conforme seu site oficial (LUA SITE OFICIAL, 2017), *Lua* é uma linguagem de programação eficiente e leve, projetada para estender aplicações. Ela é normalmente descrita como uma linguagem de múltiplos paradigmas, entre eles a programação procedural, programação orientada a objetos, programação funcional, programação orientada a dados e descrição de dados e é atualmente a linguagem de *script* mais usada em jogos.

De acordo com (MAMEdev, 2018), a integração da linguagem *Lua* com o *MAME* apareceu inicialmente na versão 0.148 do *MAME*, com uma implementação mínima do *Lua*, ainda com poucos recursos. Nas versões mais atuais a interface *Lua* já está mais desenvolvida, permitindo diversas interações com o emulador, como acesso aos registros da CPU, leitura e gravação da memória, inspeção e manipulação do estado dos dispositivos, desenhar interfaces personalizadas compostas por linhas, caixas e texto na tela do jogo, entre tantas outras funções. Apesar do fato de que a API *Lua* ainda não está completa e nem declarada como estável no *MAME*, usuários já criaram demonstrações impressionantes do que é possível fazer com essa ferramenta. O usuário grunt2084 (2017) do site github¹ foi capaz de criar um *script* em *Lua* para automatizar o clássico jogo de arcade Robotron 2084 de forma inteligente, com regras como disparar no inimigo mais próximo, escapar dos obstáculos e salvar os humanos. Para isso ser possível, foi necessário realizar uma detalhada engenharia reversa na ROM do jogo, com a finalidade de adquirir informações sobre os endereços de memória, conteúdo de cada objeto, saber onde tudo está na tela e como inserir os comandos.

2.3 Bibliotecas

Conforme descreve Magnun (2012), biblioteca, no contexto da computação, é uma coleção de funções e definições escritas para um propósito definido, e é amplamente utilizada no desenvolvimento de software, onde sua maior vantagem está em reduzir a necessidade de reescrever códigos que já foram desenvolvidos e

¹ Disponível em <<https://github.com/>>

disponibilizados. O código das bibliotecas costuma ser projetado de uma forma que permita sua utilização por outros programas. Diversas bibliotecas são fornecidas pelas próprias linguagens de programação como, por exemplo, bibliotecas que disponibilizam funções matemáticas (*math header* para C, *math* para Python, *math.js* para Javascript), e também há bibliotecas fornecidas pelos sistemas operacionais, como por exemplo os arquivos de biblioteca de vínculo dinâmico disponibilizados no *Windows* com a extensão DLL (*Dynamic-link library*) que possibilitam, por exemplo, a execução de funções do sistema operacional de baixo nível para gerenciamento de memória e manipulação de recursos (e.g., *Kernel32.dll*).

3. Trabalhos Relacionados

Neste capítulo são apresentados os trabalhos encontrados na revisão de literatura relacionados ao tema proposto. Diversas bases de busca foram utilizadas, entre elas a *Scopus*, *Institute of Electrical and Electronics Engineers (IEEE)* e *Google Scholar*. Foram encontrados vários trabalhos que expõem suas próprias plataformas para a criação de inteligência artificial para jogos, cada qual com suas limitações, pontos fortes e fracos, foco e contribuições diferente das demais. A seguir, mais detalhes sobre cada um desses trabalhos.

No trabalho (BELLEMARE et al., 2015), os autores apresentam o *Arcade Learning Environment (ALE)*, um *framework* projetado para facilitar o desenvolvimento de agentes de inteligência artificial para jogos de Atari 2600, construído em cima do emulador Stella e que fornece um ambiente de testes para avaliar e comparar diferentes abordagens. Além disso, também é realizado um *benchmark* utilizando agentes independentes de domínio com técnicas de IA de aprendizagem por reforço e algoritmos de planejamento em 55 jogos diferentes. Baseando-se nos resultados, os autores concluem que os jogos de Atari apresentam um domínio desafiador, mas não intratável, com potencial para auxiliar o desenvolvimento e a avaliação de agentes de propósito geral.

Em (JOHNSON et al., 2016), os autores expõem o projeto Malmo, uma plataforma de experimentação de inteligência artificial criada especificamente para o jogo *Minecraft*, com a intenção de auxiliar pesquisas em subáreas da IA como robótica, visão computacional, aprendizado por reforço, planejamento e sistemas multiagentes. No trabalho também é apresentada uma demonstração das capacidades do Malmo, mostrando diversas classes que podem ser usadas em diferentes linguagens de programação, com funções como geração e distribuição de tarefas para os agentes, *log* de informações, observação do ambiente, execução de ações e até mesmo uma classe que suporta missões com interação humana, para completar tarefas que sejam desafiadoras demais para tecnologias atuais de IA.

Uma linguagem de descrição simples e de alto nível para jogos 2D chamada de *Video Game Definition Language (VGDL)* é apresentada em Schaul (2013), acompanhado de uma biblioteca em *Python* criada pelo próprio autor (*PyVGDL*) que permite instanciar e interagir com tais descrições. As descrições dos jogos são definidas por dois componentes, a descrição do nível, que mapeia as posições

iniciais de todos os objetos do jogo em formato textual, e a descrição do jogo, que especifica as dinâmicas e interações de todos os objetos do jogo, sendo essa última dividida em quatro blocos de instruções. Na Figura 3 é demonstrado um exemplo de descrição de nível para um jogo estilo *The Legend of Zelda*¹, onde 'A' representa o herói, '+' a chave, '1' os monstros e 'w' as paredes, descrito de forma textual à esquerda e renderizado pela biblioteca à direita.

Figura 3 - Descrição de nível em VGDG



Fonte: (SCHAUL, 2013)

Para demonstrar o espectro de jogos que podem ser codificados no PyVGDL e também para servir como exemplos para desenvolvedores, o autor implementou versões simplificadas de diversos jogos clássicos, como *Space Invaders*, *Pac-Man*, *Lunar Lander* e *Super Mario*, todos fornecidos juntos com a biblioteca.

Já o trabalho (PEREZ-LIEBANA et al., 2016) apresenta um *framework* para a criação de agentes inteligentes para jogar, bem como uma competição em que o objetivo principal é a criação de inteligências artificiais capazes de jogar diversos jogos - ambos compartilham o nome de *General Video Game AI (GVGAI)*. O *framework* em questão difere do ALE citado anteriormente pois, ao invés de utilizar a captura de tela para extrair o estado do jogo, fornece as informações de forma orientada a objetos e usa a linguagem de descrição *VGDL*, criada por Schaul (2013), para definir os jogos. Isso significa que o *framework* não está limitado a apenas um jogo como o *Malmo* (com *Minecraft*), ou à uma plataforma como o *ALE* (com o Atari 2600), pois é possível manipular qualquer jogo que esteja descrito em *VGDL*. No momento de publicação do artigo em questão já haviam cerca de 80 jogos.

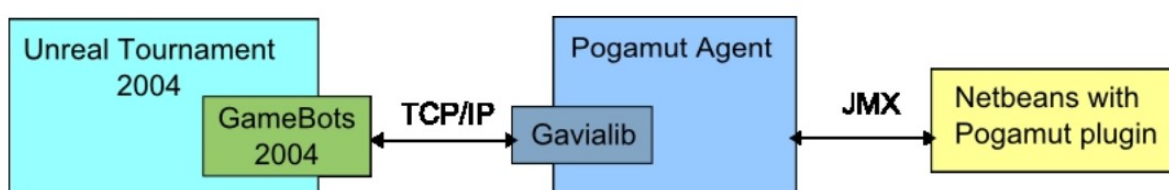
Em (BÍDA et al., 2012) é apresentado o projeto *GameBots* e suas evoluções. O projeto *GameBots* tem como objetivo principal fornecer aos pesquisadores um

¹ Jogo de ação e aventura criado pela Nintendo para o console NES, em 1986

ambiente de teste virtual em tempo real para seus agentes, possibilitando a integração dos agentes com os jogos da série *Unreal Tournament*. Para isso foi realizada uma modificação no jogo, através da linguagem de *scripts* disponibilizada pelo mesmo, chamada de *UnrealScript*. Essa modificação permite que os personagens sejam controlados através de *sockets* TCP/IP, provendo as informações sensoriais do ambiente para o programa cliente através da conexão de rede. Após decidir as ações que devem ser executadas, o programa cliente envia os comandos para o jogo pela rede. O trabalho cita diversos projetos que foram criados utilizando o *GameBots* como base, destacando o *Pogamut*, o qual diferencia-se dos demais, uma vez que otimizou seu desempenho e incorporou vários recursos na forma de uma plataforma que oferece ferramentas de codificação para agentes virtuais, sendo usada, inclusive, na competição *BotPrize* (HINGSTON, 2009).

O artigo (GEMROT et al., 2009) tem como objetivo apresentar a versão atualizada do *Pogamut* citado no trabalho anterior, chamado de *Pogamut 3*, bem como identificar seus novos recursos, arquitetura e extensões. O *Pogamut 3* pode ser considerado um *kit* de ferramentas de desenvolvimento para agentes inteligentes para video games e uma de suas características mais importantes é que ele foi projetado tanto para pesquisadores experientes quanto para iniciantes na área, já havendo sido testado como ferramenta de ensino em uma universidade e em um curso de ensino médio. O *Pogamut 3* é composto pela integração de 5 componentes, o jogo *Unreal Tournament 2004*, *GameBots2004*, biblioteca *GaviaLib*, o agente *Pogamut* e o ambiente de desenvolvimento integrado com o *NetBeans* na forma de um *plugin*, que se comunica com os agentes via *JMX* (*Java Management Extensions*). Uma representação da arquitetura do *Pogamut 3* pode ser observada na Figura 4.

Figura 4 - Arquitetura do Pogamut 3



Fonte: (GEMROT et al., 2009)

Em seu trabalho, Bhonker et al. (2016) apresentam a plataforma *Retro Learning Environment (RLE)*, um ambiente de aprendizagem inspirado no ALE, com a diferença de que o RLE foi projetado para suportar jogos de diversos consoles diferentes, não apenas o Atari 2600. Isso é possível devido a abordagem implementada, que separa a camada do ambiente de aprendizagem do emulador, ao contrário do ALE em que o ambiente é acoplado ao emulador. Para isso foi utilizada a interface LibRetro, que auxilia na comunicação entre programas e emuladores. Apesar de virtualmente ser capaz de integrar mais de 7000 jogos, de diversos consoles, no momento de publicação do artigo a plataforma oferecia suporte para apenas 8 jogos. Isso acontece, pois, para cada novo jogo suportado, é necessário criar dois novos arquivos de código fonte e recompilar o RLE.

Outro ambiente de aprendizagem inspirado no ALE é apresentado por Dubrovsky (2017). Chamado de *MAME Learning Environment (Mamele)*, o projeto consiste em uma bifurcação ou ramificação (do inglês *fork*) do repositório principal do emulador *MAME* para permitir a criação de agentes inteligentes para jogos de *arcade*. Nenhuma publicação sobre essa ferramenta foi encontrada e tampouco foi possível testá-la, uma vez que não foi possível compilar o projeto, mesmo depois de seguir à risca os passos encontrados na sua documentação oficial. Devido à semelhança de proposta entre a plataforma *Mamele* e a plataforma proposta neste trabalho, decidimos priorizar, como diferencial, a facilidade de utilização para o projeto, evitando que haja a necessidade de compilar o código para utilizar a plataforma, como no *Mamele*, ou para adicionar novos jogos, como no *RLE*.

4. Metodologia

Neste capítulo é apresentada a metodologia utilizada para o desenvolvimento do trabalho, descrevendo as atividades, procedimentos e ferramentas que foram adotados para alcançar os objetivos do trabalho.

Inicialmente foram realizadas pesquisas online, estudos e implementação de protótipos para definir que tecnologias seriam trabalhadas, linguagem de programação, emulador, etc.

Foi necessário realizar uma análise mais aprofundada na arquitetura do emulador selecionado, para então decidir como seria construída a biblioteca e de que forma ela iria interagir com os comandos do emulador. Também foi necessário investigar formas para a obtenção dos quadros dos jogos em tempo de execução eficientemente. Para ambos problemas foi decidido utilizar bibliotecas *DLL* do *Windows* como solução, que se mostrou eficaz tanto para interagir com os comandos do emulador quanto para obter os quadros do jogo.

Para obter os dados que foram utilizados nos *scripts* dos experimentos em *Lua*, foi utilizada uma estratégia semelhante ao programa *Cheat Engine*¹, que é utilizado para ler e escrever dados diretamente na memória. Este programa costuma ser usado para “trapacear” em jogos de computador mas, caso tivéssemos optado por adotar essa mesma estratégia, a biblioteca estaria restrita a ler os dados da memória para utilizá-los nos *scripts*, sem ter como foco alterar seus valores.

Além dessa estratégia, também testamos os *scripts* considerando os *frames* do jogo sendo executado no emulador como entrada para o seu processamento, trabalhando diretamente com as imagens do jogo como parâmetros para realizar suas decisões. Baseado nos resultados apresentados com mais detalhes no capítulo a seguir, foi decidido prosseguir com essa estratégia, que se mostrou de mais fácil utilização para o usuário.

Após a definição das tecnologias e o seu aprendizado, foi iniciado o projeto e a implementação da biblioteca propriamente dita, utilizando um método semelhante à metodologia de desenvolvimento ágil e o modelo de processo de desenvolvimento iterativo e incremental.

Por fim, foi elaborada uma documentação para orientar o uso da biblioteca, onde foram apresentados exemplos de sua utilização, através da aplicação de

¹ Disponível em <<https://www.cheatengine.org/>>

algoritmos clássicos da IA. Os exemplos servem tanto para testar a biblioteca quanto para ilustrar seu modo de uso para os alunos, professores e pesquisadores.

Através da Tabela 1 é possível visualizar o cronograma das atividades que foram executadas ao longo do desenvolvimento do trabalho.

Tabela 1 - Cronograma de Atividades

Atividades	Mar	Abr	Mai	Jun	Jul	Ago	Set	Out	Nov	Dez
Escolha do tema	X									
Pesquisa bibliográfica	X	X	X							
Revisar e analisar a referência bibliográfica	X	X	X							
Escolha das tecnologias (linguagem, emulador,...)		X								
Estudo das tecnologias		X	X							
Elaboração do projeto		X	X	X						
Entrega do projeto (apresentação TCC I)				X						
Desenvolvimento da solução proposta				X	X	X	X	X		
Testes finais								X	X	
Redação da monografia					X	X	X	X	X	
Revisão e entrega do trabalho									X	
Apresentação do TCC II										X

4.1 Software

A implementação dos testes foi realizada utilizando a linguagem de programação *Python*, versão 3.7.0 com as seguintes bibliotecas:

- *Pillow* 5.2.0
- *OpenCV*
- *pywin32* 224
- *OpenCV-Python* 3.4.3.18
- *Keyboard* 0.13.2
- *Numpy* 1.15.2

O emulador utilizado foi o *MAME*, versão 0.193. É necessário alterar o arquivo de configuração *mame.ini*, definir “*dinput*” como novo valor para a propriedade *keyboardprovider*.

A Rom utilizada para os testes foi *Super Tank*¹, que pode ser encontrada de forma livre no site oficial do *MAME*.

¹ Disponível em: <<https://www.mamedev.org/roms/supertnk/>>

4.2 Hardware

Os experimentos foram executados em uma máquina com as seguintes configurações:

- Processador: Intel Core i7-2600K
- Memória: 8GB DDR3
- Sistema Operacional: Windows 10 64bits

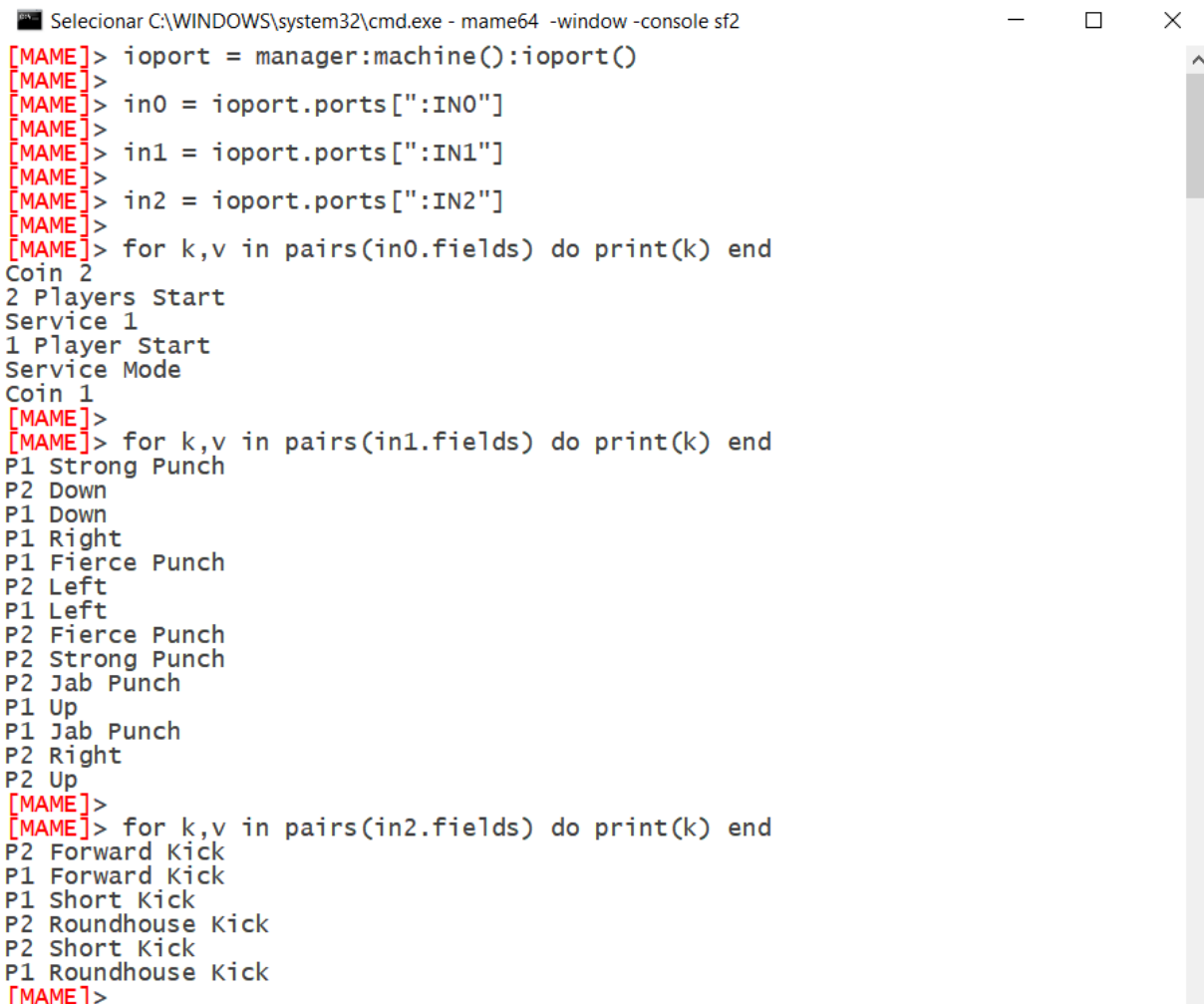
5. Desenvolvimento do Trabalho

Neste capítulo é apresentado o desenvolvimento da biblioteca propriamente dita, incluindo a fase de definição de ferramentas, analisando qual é a linguagem mais eficiente para a criação da biblioteca, a arquitetura utilizada e as funções implementadas.

5.1 Definição das ferramentas utilizadas

Primeiramente, foram realizados testes com a integração do *MAME* com *Lua*, para avaliar a possibilidade de utilizá-la como base para o projeto. Tendo como base o *script* criado por *grunt2084* para automatizar jogo *Robotron 2084*, foi desenvolvido um novo *script* de automação, para o jogo *Street Fighter 2*, com a intenção de entender e testar a forma de interação entre os comandos desejados e o emulador. O mapeamento dos comandos é único para cada máquina, o que inviabiliza a criação direta de algoritmos para múltiplos jogos, pois para cada novo jogo haveria a necessidade de investigar como é feito o mapeamento dos comandos de entrada. Dessa forma, fica evidente a importância do presente trabalho, com a elaboração de uma alternativa para a criação de *scripts* universais, permitindo seu uso com pouca ou nenhuma alteração em diferentes jogos. Na Figura 5 é possível observar como é realizado o mapeamento de botões do jogo *Street Fighter 2*.

Figura 5 - Mapeamento de botões de um jogo de arcade



```

Selecionar C:\WINDOWS\system32\cmd.exe - mame64 -window -console sf2
[MAME] > ioport = manager:machine():ioport()
[MAME] >
[MAME] > in0 = ioport.ports[":IN0"]
[MAME] >
[MAME] > in1 = ioport.ports[":IN1"]
[MAME] >
[MAME] > in2 = ioport.ports[":IN2"]
[MAME] >
[MAME] > for k,v in pairs(in0.fields) do print(k) end
Coin 2
2 Players Start
Service 1
1 Player Start
Service Mode
Coin 1
[MAME] >
[MAME] > for k,v in pairs(in1.fields) do print(k) end
P1 Strong Punch
P2 Down
P1 Down
P1 Right
P1 Fierce Punch
P2 Left
P1 Left
P2 Fierce Punch
P2 Strong Punch
P2 Jab Punch
P1 Up
P1 Jab Punch
P2 Right
P2 Up
[MAME] >
[MAME] > for k,v in pairs(in2.fields) do print(k) end
P2 Forward Kick
P1 Forward Kick
P1 Short Kick
P2 Roundhouse Kick
P2 Short Kick
P1 Roundhouse Kick
[MAME] >

```

Fonte: Elaborado pelo autor.

Após os testes realizados com a linguagem *Lua*, foi decidido testar a viabilidade do desenvolvimento da biblioteca em *Python*, linguagem escolhida por ser a mais utilizada pela comunidade de inteligência artificial e possuir diversas bibliotecas que poderão ser utilizadas em conjunto com a proposta deste trabalho, além de ser relativamente simples e de fácil aprendizado, correspondendo aos objetivos do projeto. Para realizar tais testes, foi estipulada a criação de uma interface simples com o emulador *MAME*, para recriar o *script* de automação para o jogo *Street Fighter 2*, mas agora em *Python*. A estratégia utilizada para a criação da interface foi simular eventos de teclado através de bibliotecas já existentes para esta funcionalidade em *Python*, porém, apesar dos eventos de pressionamento de teclas funcionarem na maioria das aplicações, não era possível interagir com o *MAME*. Após extensa análise no código-fonte do *MAME*, foi possível entender como forçar o emulador a aceitar eventos de teclado de forma virtual, bem como o mapeamento

correto das teclas, por meio do arquivo “*input_common.h*”.

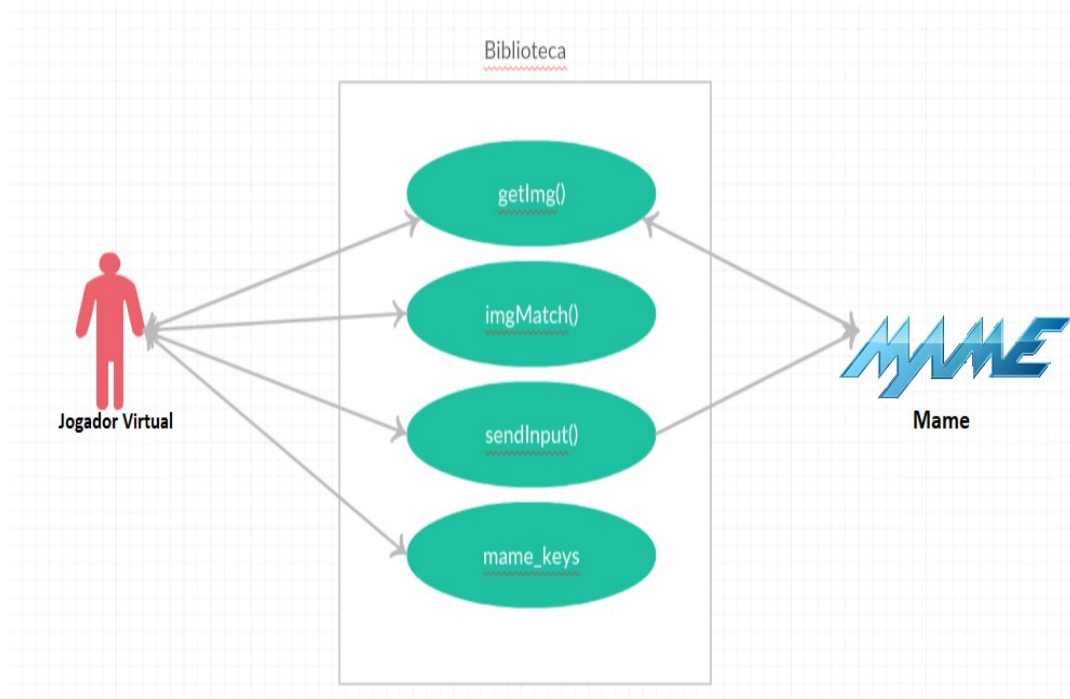
Podemos dizer que os experimentos em *Python* obtiveram êxito pois, foi possível recriar fielmente o comportamento do *script* de automação em Lua para o jogo *Street Fighter 2* utilizando a biblioteca desenvolvida para servir como interface entre o *MAME* e os comandos determinados no *script*. Também foi possível perceber a vantagem de utilizar a simulação de eventos de teclado em comparação com a forma de entrar com os inputs em lua, pois não há mais a necessidade de rastrear o mapeamento de cada jogo como explicado anteriormente, bastando informar as teclas do teclado que deverão ser pressionadas.

Apesar dos resultados positivos, ainda há muito a fazer para contemplar todos os objetivos propostos neste trabalho, como por exemplo realizar melhorias na biblioteca para encurtar os comandos de pressionamento de teclas e adicionar a possibilidade de visualizar os *frames* do jogo.

5.2 Arquitetura da Biblioteca

Primeiramente foram definidas as funções fundamentais que a biblioteca precisa entregar ao desenvolvedor para garantir que seja possível criar um *script* de inteligência artificial para o emulador *MAME*. Para isso foi criado um diagrama contendo as funções mínimas exigidas pela biblioteca, que poderão ser usadas pelo jogador virtual criado pelo desenvolvedor, como demonstra a Figura 6.

Figura 6 – Funções Principais



Fonte: Elaborado pelo autor.

Como as funções exigidas na criação da biblioteca são bem definidas e independentes, foi decidido separar a biblioteca em módulos distintos, cada qual com a responsabilidade de atender um assunto bem delimitado, podendo, inclusive, ser utilizado independentemente dos outros módulos. Essa abordagem oferece vantagens como legibilidade, facilidade na manutenção do código e flexibilidade para novas implementações, visto que, da forma como os módulos foram projetados, é possível futuramente estender a biblioteca para outros emuladores e até mesmo jogos específicos.

A biblioteca criada para realizar os experimentos na seção de definição de ferramentas foi dividida em dois módulos diferentes, um para efetuar a injeção de sinais de controle e o outro para armazenar o mapeamento das teclas do *MAME*. Também foi adicionado um terceiro módulo, para possibilitar a captura da tela do jogo e a detecção de objetos através de imagens de modelo (*template*).

5.2.1 Módulo de simulação de eventos de teclado

A principal função desse módulo é enviar de sinais de eventos de teclado para o *MAME*. Para realizar essa função da maneira correta foi necessário instanciar

arquivos *DLL* do *Windows* através da biblioteca *ctypes*¹, pois, no *Windows*, a forma mais abrangente de simular o pressionamento de botões do teclado e enviar sinais de teclado e *mouse* para o sistema operacional é através dos arquivos *DLL* do próprio sistema. Existem diversas outras bibliotecas para *Python* que conseguem simular *inputs* do usuário, como por exemplo a biblioteca *PyAutoGUI*². Contudo, o sinal gerado por essas bibliotecas não são compreendidos por grande parte das aplicações, incluindo grande parte dos jogos e o emulador *MAME*, que só percebem *inputs* virtuais se estiverem na estrutura de "*Direct Input*"³, o que é possível no *Windows* apenas com a solução adotada, acessando funções das *DLLs* carregadas pela biblioteca *ctypes*.

5.2.2 Módulo de mapeamento de teclas

Esse módulo contém o mapeamento de todas as teclas utilizadas pelo *MAME*, transpondo para *Python* todo o mapeamento de entrada apresentado pelo código-fonte do *MAME* por meio do arquivo *input_common.h*⁴. A decisão de manter o mapeamento de teclas em um arquivo separado foi motivada pela facilidade com que esse tipo de implementação permite que a biblioteca se adapte aos comandos de outros jogos e emuladores, caso necessário em uma versão futura.

5.2.3 Módulo de captura de tela

Esse módulo é o responsável pela captura dos *frames* do jogo que está sendo executado pelo *MAME*, em tempo real, para que as imagens do jogo possam ser levadas em conta para auxiliar nas decisões dos algoritmos. Para garantir a obtenção da tela do jogo é necessário certificar-se que o processo do *MAME* está aberto e em primeiro plano, para isso foi necessário utilizar funções da biblioteca *win32gui*⁵. Foi decidido incluir também a biblioteca *OpenCV*⁶ ao módulo, para tornar mais acessível a utilização de métodos de visão computacional, retornando na captura de imagens, por padrão, a imagem no formato utilizado pelo *OpenCV*, com a

¹ Disponível em <<https://docs.python.org/3/library/ctypes.html>>

² Disponível em <<https://pyautogui.readthedocs.io/>>

³ Disponível em <[https://docs.microsoft.com/en-us/previous-versions/windows/desktop/ee418273\(v%3dvs.85\)](https://docs.microsoft.com/en-us/previous-versions/windows/desktop/ee418273(v%3dvs.85))>

⁴ Disponível em <https://github.com/mamedev/mame/blob/master/src/osd/modules/input/input_common.h>

⁵ Disponível em <<https://pypi.org/project/win32gui/>>

⁶ Disponível em <<https://opencv.org/>>

finalidade de poder utilizar qualquer método de imagens encontradas no mesmo, como por exemplo a função *matchTemplate*, que serve para detectar objetos em uma imagem. Tal função utiliza diversos parâmetros, então, para facilitar o trabalho do desenvolvedor, foram criadas versões facilitadas de rotinas de detecção de objetos, cada uma com um objetivo diferente e com os parâmetros otimizados para melhor desempenhar sua função nos jogos do *MAME*, sendo necessário enviar como parâmetro apenas a imagem original e o template que será procurado dentro dela.

5.3 Apresentação e Testes das Funções

Nesta seção serão apresentadas as funções dos módulos bem como os testes que foram realizados nas mesmas, em tempo de desenvolvimento, para garantir que os objetivos de cada função estejam plenamente atendidos.

5.3.1 Função de Simulação de Eventos de Teclado

Os testes de simulação de eventos de teclado foram realizados no jogo *Street Fighter 2*, com um algoritmo simples de automação que apenas pressiona os comandos de forma aleatória, apenas para certificar se o emulador *MAME* está recebendo os sinais de forma correta. O *software* se comportou da maneira esperada e todos os eventos foram interpretados de modo correto pelo emulador, tanto no jogo *Street Fighter 2* quanto no jogo *Super Tank*, executando de maneira correta todos os comandos do jogador 1.

Para utilizar esta função basta informar a tecla que deve ser utilizada, utilizando o mapeamento correto, e qual o evento vai ser realizado, caso não seja enviado o segundo parâmetro, por padrão será considerado o evento de pressionamento de tecla, para liberar a tecla o evento que deve ser enviado é o *KEYEVENTF_KEYUP*. Também é possível enviar um vetor contendo mais de um comando, para que sejam realizados simultaneamente. O código para demonstração de como utilizar essa função pode ser visto no Apêndice B.

5.3.2 Função de Captura de Tela

Os testes de captura de tela consistem em confirmar se a função é capaz de entregar o *frame* atual do jogo e validar se o comportamento está adequado no caso

do emulador não estar aberto. Para realizar esse teste foi utilizado o jogo *Super Tank*, distribuído livremente no site oficial do MAME.

A função retorna corretamente a tela do jogo caso o processo esteja aberto no sistema, e caso o processo não esteja em primeiro plano, a própria função força para que ele fique em primeiro plano, e só retorna a imagem quando isso acontece. Há, porém, algumas ressalvas quanto ao seu uso, não sendo possível obter a imagem caso o emulador esteja em tela cheia, e é importante perceber que a imagem obtida pela função contém não apenas o *frame* do jogo, como também o entorno da imagem, como pode ser visto na Figura 7.

Figura 7 – Captura de Tela



Fonte: Elaborado pelo autor.

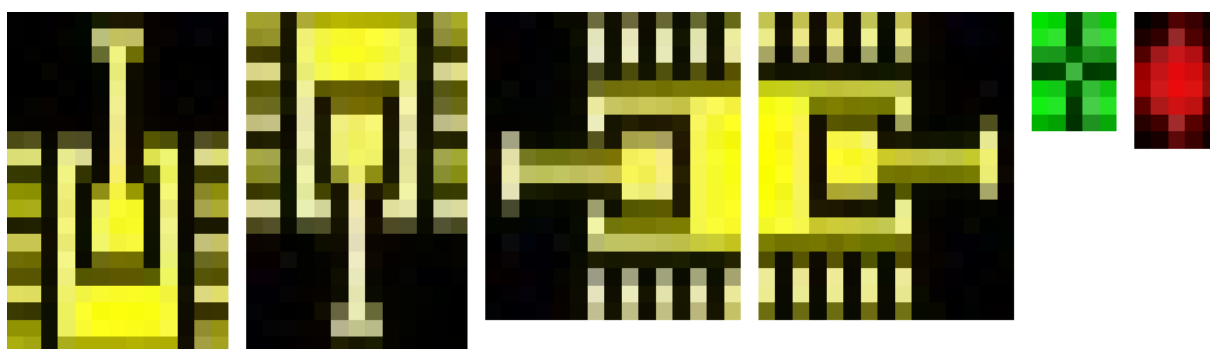
A verificação da mensagem no caso do emulador não estar aberto ocorreu conforme o esperado, sem retorno de erros, apenas exibindo a mensagem “Por favor abra o *MAME* para continuar.”. O código para demonstração de como utilizar essa função pode ser visto no Apêndice B.

5.3.3 Função de Detecção de Objetos

Foram realizados diversos testes com a função de detecção de objetos,

principalmente utilizando o jogo *Super Tank*. A função básica de detecção utiliza apenas um canal de cor, para garantir a entrega mais rápida possível do resultado. O retorno da função é um vetor contendo as coordenadas do canto superior esquerdo em todas as localizações em que a imagem de *template* foi encontrada, e um vetor vazio caso o *template* não esteja contido na imagem. Na Figura 8 é possível ver uma versão ampliada de todos os *templates* que foram utilizados para testar a função no jogo *Super Tank*.

Figura 8 – Templates para o jogo *Super Tank*

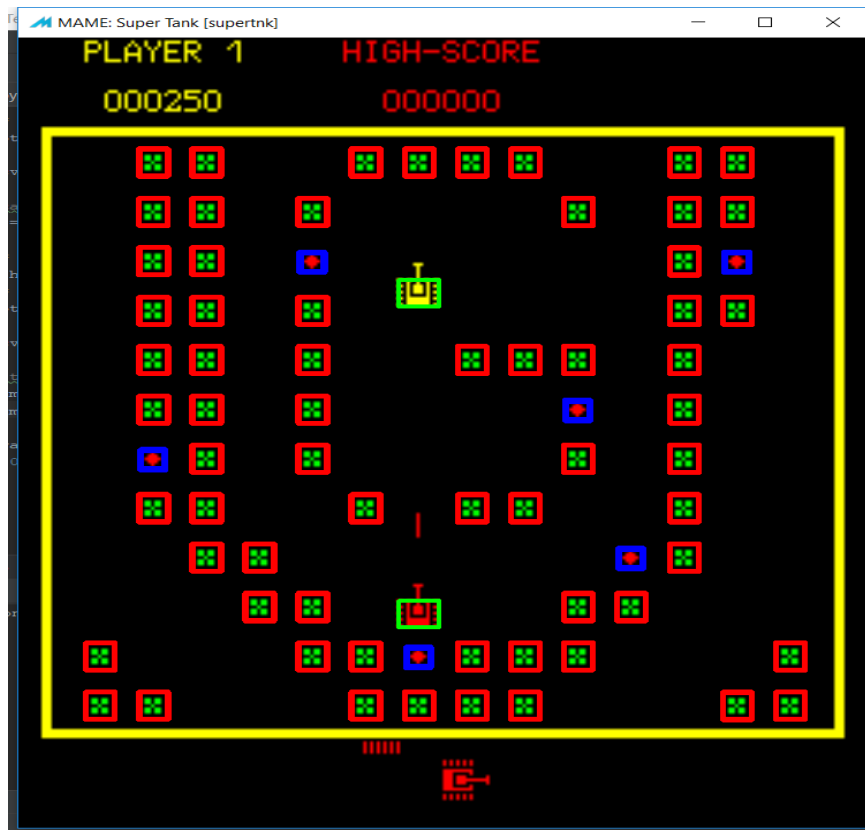


Fonte: Elaborado pelo autor.

Para criar o *template* basta abrir a tela que contenha o formato que deseja detectar, pressionar a tecla *Print Screen*, abrir o programa de edição de imagens (*Paint* ou *GIMP* por exemplo), colar a imagem pressionando as teclas *Ctrl + V*, recortar a parte que deseja utilizar de *template* e salvar no disco rígido, utilizando de preferência os formatos *PNG*, *BMP* ou *JPG* (os dois primeiros são preferíveis pois não alteram o conteúdo das cores da imagem). No código, é preciso fornecer o caminho do arquivo com a forma a ser detectada através do comando *imread* do *OpenCV*. O código para demonstração de como utilizar essa função pode ser visto no Apêndice B.

Para identificar mais facilmente o resultado encontrado pela função, decidimos destacar as ocorrências das coordenadas na imagem original para cada objeto encontrado, com a cor da borda identificando o tipo de objeto pesquisado, o resultado pode ser visto na Figura 9.

Figura 9 – Detecção de Objetos



Fonte: Elaborado pelo autor.

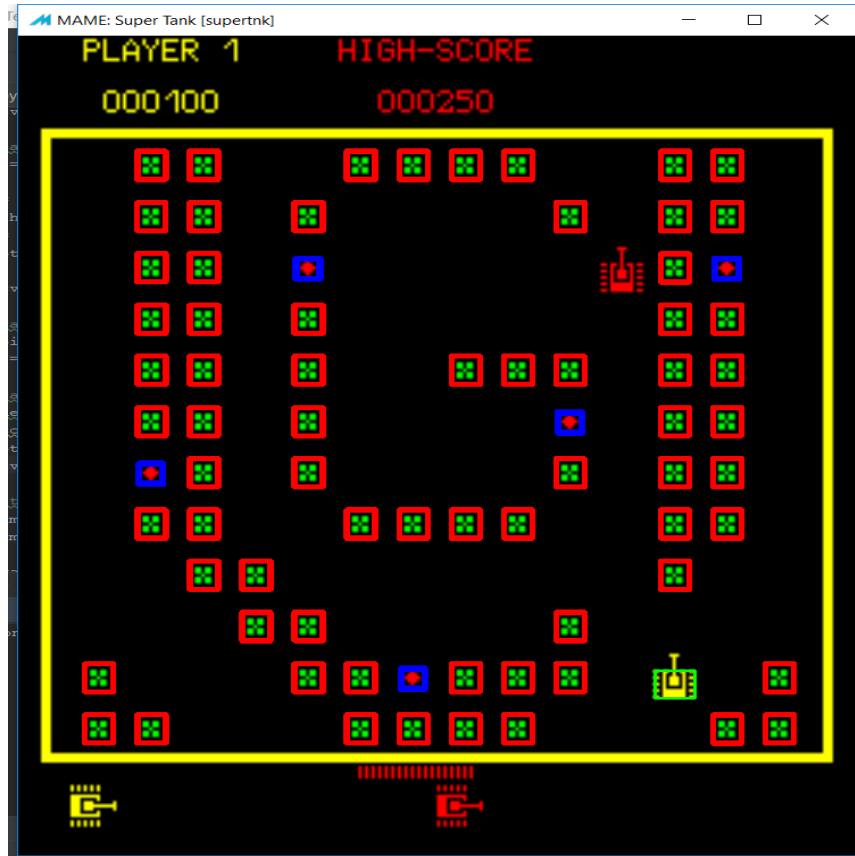
O objetivo do algoritmo é destacar com um retângulo vermelho os objetivos verdes, destacar com um retângulo azul os objetivos azuis e destacar com um retângulo verde o tanque amarelo, que é controlado pelo jogador. É possível perceber que todos os objetos foram encontrados e destacados corretamente, com exceção dos tanques, pois a imagem pesquisada era para retornar apenas a coordenada do tanque amarelo, no entanto o tanque vermelho também foi encontrado. Verifica-se que isso ocorre pelo fato do método utilizado considerar apenas um canal de cor, o que garante velocidade mas não consegue diferenciar objetos iguais com cores diferentes. Para contornar esse problema foi criado o método a seguir.

5.3.4 Função de Detecção de Objetos com Cor

Função criada para resolver o problema encontrado na detecção padrão, que não leva em consideração as cores da imagem. Para testar foi executado o mesmo teste da detecção padrão, utilizando o jogo *Super Tank*. A rotina retorna corretamente todas coordenadas, levando em consideração a forma e também a cor

das imagens, encontrando o tanque correto em qualquer posição, como mostra a Figura 10 e o código para demonstração de como utilizar essa função pode ser visto no Apêndice B.

Figura 10 – Detecção de Objetos com Cor



Fonte: Elaborado pelo autor.

5.3.5 Função de Detecção de Objetos com Transparência

Como em grande parte dos jogos há um cenário de fundo, que pode acabar atrapalhando a detecção de objetos e personagens através dos métodos normais, foi criada uma função de detecção que leva em conta a transparência da imagem, ignorando a cor informada na imagem de template ao fazer a comparação com a imagem original. Como o jogo *Super Tank* não possui cenário de fundo, os testes foram realizados no jogo *Street Fighter 2*, onde o personagem só era encontrado quando estava localizado exatamente no mesmo local da imagem de template se fosse utilizada a função padrão. Para utilizar a função é necessário remover do template todos os detalhes que devem ser ignorados, pintando-os de branco, como foi feito na Figura 11.

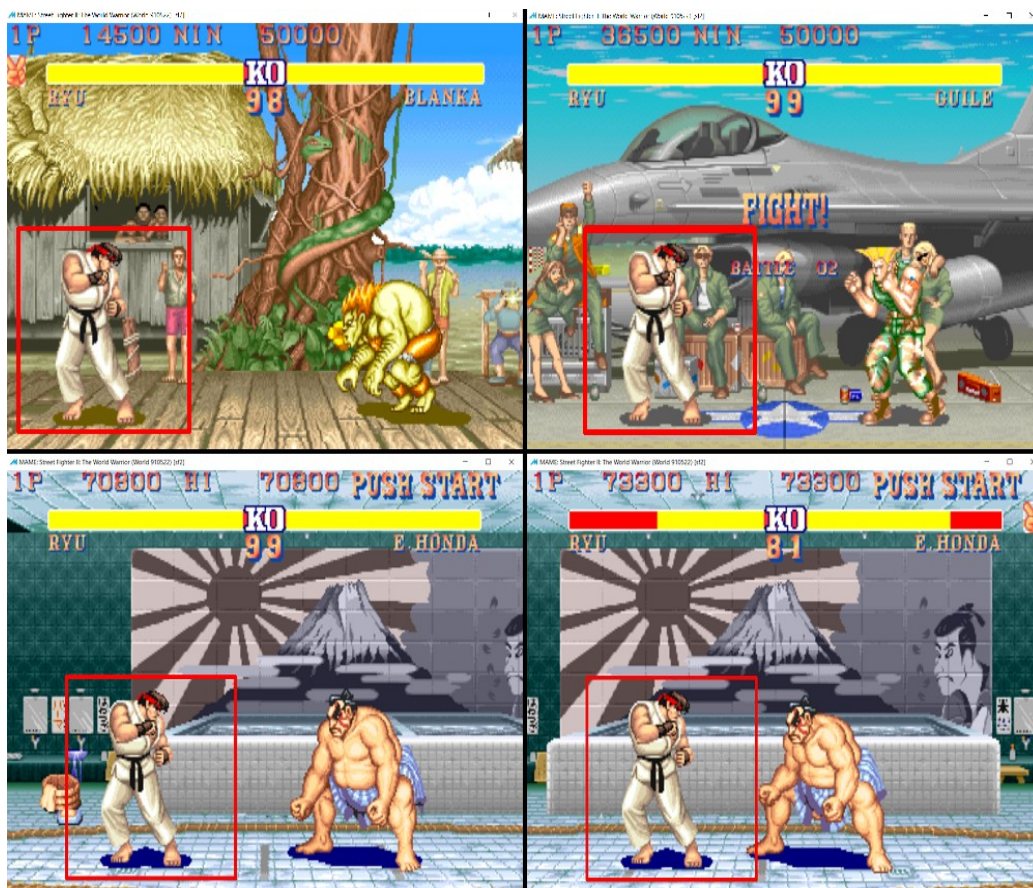
Figura 11 – Template para função de detecção com transparência



Fonte: Elaborado pelo autor.

Caso o novo *template* fosse utilizado na função normal de detecção, o personagem não seria encontrado em lugar algum, pois seria necessário que houvesse um cenário todo branco para que isso acontecesse. Para a transparência ser levada em conta é utilizada uma máscara no *template* e os parâmetros da função *matchTemplate* precisaram ser alterados, porém, para o desenvolvedor, basta chamar a função com transparência da mesma forma que chama as outras, enviando apenas a imagem original e o template. Os resultados podem ser vistos na Figura 12 e o código para demonstração de como utilizar essa função pode ser visto no Apêndice B.

Figura 12 – Detecção de Objetos com Transparência



Fonte: Elaborado pelo autor.

Percebe-se que a função foi capaz de encontrar o personagem, independentemente do cenário em que ele se encontra.

6. Resultados

Neste capítulo será mostrado qual foi o ambiente de teste utilizado e como foi realizada a experiência para validar a biblioteca, como a corretude das funções já foi testada anteriormente, esse capítulo focará no funcionamento da integração dos módulos da biblioteca como um todo.

6.1 Algoritmo

Foi criado um algoritmo de automação para o jogo Super Tank, utilizando as principais funções da biblioteca, com uma estratégia simples, com o intuito de avaliar a integração dos módulos da biblioteca e a facilidade de uso.

Primeiramente o algoritmo detecta a localização dos objetivos e do tanque controlado pelo jogador, depois é calculada a distância entre o jogador e todos os objetivos para saber qual é o objetivo mais próximo do jogador, e por fim a estratégia consiste basicamente em andar na direção do objetivo mais próximo, considerando, a princípio, apenas os objetivos vermelhos e, caso não haja mais objetivo vermelho na tela, passa a considerar os verdes também. Na criação do algoritmo foi certificada a viabilidade de uso da biblioteca, bastando realizar a sua importação para ter acesso à todas as funções apresentadas.

6.2 Execução na Resolução Padrão do MAME

O primeiro teste foi feito na resolução entregue por padrão pelo *MAME*, que redimensiona a imagem original do jogo para oferecer uma melhor exibição em computadores mais modernos, uma vez que a resolução dos jogos antigos costumava ser muito baixa.

A biblioteca comportou-se conforme o esperado, entregando as imagens, detectando os objetos e executando os comandos decididos pelo algoritmo, executando em média 220 iterações por minuto. No entanto, notou-se que em alguns momentos a decisão de troca de direção demorava a acontecer, o que motivou a execução de uma segunda rodada de testes, sem o redimensionamento de resolução do MAME.

6.3 Execução na Resolução Nativa do Jogo

Para realizar esse teste foi necessário recortar as imagens novamente, para que ficassem de acordo com a resolução nativa do jogo, que é de 456x262 pixels.

O ganho de performance foi considerável, conseguindo agora executar em média 500 iterações por minuto, o que tornou o algoritmo mais estável e confiável. Além dessa vantagem, vale a pena constar que a resolução padrão do *MAME* varia de computador para computador, enquanto a resolução nativa do jogo é fixa, ou seja, um algoritmo que leve em consideração a resolução nativa do jogo tende a funcionar e ser replicável em praticamente todos os sistemas que atendam aos requisitos para executar o emulador e as tecnologias utilizadas pela biblioteca.

7. Considerações Finais

Após transcorrido o desenvolvimento do trabalho e dos testes realizados, concluiu-se que o desenvolvimento de uma biblioteca para substituir a atuação de um jogador no emulador *MAME*, que fosse utilizável na prática, capaz de entregar e alcançar os objetivos gerais propostos nesse trabalho foi concluído com sucesso. Dessa forma, os objetivos originais, que eram a criação de uma biblioteca¹ que pudesse ser empregada para a utilização de algoritmos de Inteligência Artificial em conjunto com o emulador *MAME*, a qual fosse fácil de usar e que permitisse a interação completa com os jogos em tempo de execução foram plenamente atingidos.

Através dos experimentos realizados, a biblioteca demonstrou-se estável e flexível, além de disponibilizar também funções capazes de obter os *frames* dos jogos de maneira eficiente, detectar elementos em um cenário bastante dinâmico de maneira funcional e de atuar nos controles dos jogos com uma latência aceitável conforme especificado nos objetivos específicos. Também foi possível perceber a importância de se criar os algoritmos e os *templates* a serem detectados levando em consideração a resolução nativa dos jogos, sendo vantajoso tanto em termos de desempenho quanto em termos de padronização, pois o redimensionamento feito pelo emulador gera deformações e interpolações as quais interferem drasticamente na qualidade e no desempenho do algoritmo de detecção, podendo também inviabilizar totalmente o resultado da rotina de detecção no caso do template ter sido retirado em uma resolução diferente da que está sendo executada no momento. Dessa forma, torna-se disponível, para a comunidade mundial de pesquisa em IA e Aprendizagem de Máquina, uma valiosa ferramenta para investigação e validação de algoritmos inteligentes dos mais diversos tipos, os quais poderão ser utilizados em conjunto com uma grande diversidade de jogos disponíveis para o emulador *MAME*, bastando para isso seguir as instruções de instalação, configuração e utilização presentes no Apêndice A deste texto.

Pelo fato da biblioteca utilizar diversas funções de bibliotecas *DLL* (*Dynamic Link Library*) do *Windows*, atualmente ele ainda é o único sistema operacional em que a biblioteca funciona. Assim, propõe-se, em trabalhos futuros, a adaptação da biblioteca para outros sistemas operacionais como o *Linux*, que é mais popular em

¹ Disponível em: <<https://github.com/RicardoDora/MameVirtualPlayer>>

ambientes acadêmicos, por exemplo. Além disso, para possibilitar que múltiplos jogadores virtuais possam atuar simultaneamente em um mesmo jogo, é interessante paralelizar as funções da biblioteca, utilizando uma tecnologia como *CUDA*, *OpenACC* ou *OpenCL* por exemplo, acelerando também as rotinas de detecção de objetos e permitindo que múltiplos algoritmos possam gerenciar jogadores a competir ou cooperar em um mesmo jogo concomitantemente. Finalmente, pretendemos disponibilizar a biblioteca online no site *GitHub* sob o nome *MameVirtualPlayer* para que possa ser utilizada amplamente.

Referências

- AGÊNCIA O GLOBO. *Inteligência artificial ganha orçamento próprio nas empresas do país*. 2017. Disponível em: <<https://epocanegocios.globo.com/Tecnologia/noticia/2017/12/inteligencia-artificial-ganha-orcamento-proprio-nas-empresas-do-pais.html>>. Acesso em: 02 de junho 2018. Citado na página 12.
- BADHAM, Direção de John. *Jogos de Guerra*. Produção de Leonard Goldberg. Hollywood. United Artists, 1983. 1h54min. Citado na página 13.
- BAER, Michael. *Emulating the ARM Architecture Using a Java Dynamic Binary Translator*. 2007. Dissertação de Mestrado. School of Computer Science, University of Manchester. Manchester. Citado na página 17.
- BELLEMARE, M. G.; NADDAF, Y.; VENESS, J. ; BOWLING, M. *The Arcade Learning Environment: An Evaluation Platform for General Agents*. *Anais do 24º International Joint Conference on Artificial Intelligence (IJCAI 2015) (Resumo expandido)*. p. 4148-4152. 2015. Citado na página 23.
- BHONKER, N.; ROZENBERG, S.; HUBARA, I.; *Playing SNES in the Retro Learning Environment*. 2016. Preprint arXiv:1611.02205, 2016. Citado na página 12 e 26.
- BÍDA, M.; ČERNÝ, M. ; GEMROT, J.; BROM, C. *Evolution of GameBots project*. In: Herrlich, M., Malaka, R., Masuch, M. (eds.) ICEC 2012. LNCS, vol. 7522, p. 397-400. Springer, Heidelberg, 2012. Citado na página 24.
- CONLEY, J.; ANDROS, E.; CHINAI, P.; LIPKOWITZ, E.; PEREZ, D. *Use of a Game Over: Emulation and the Video Game Industry*. *Northwestern Journal of Technology and Intellectual Property*, 2(2). Illinois. 2004. Citado na página 20.
- DUBROVSKY, Alejandro. *Machine learning environment over MAME-supported games*. 2017. Disponível em: <<https://github.com/alito/mamele>>. Acesso em: 27 de maio 2018. Citado na página 26.
- ENSMENGER, N. *Is chess the drosophila of artificial intelligence? A social history of an algorithm*. *Social Studies of Science* 42 (1), p. 5–30. Sagepub, New York. 2012. Citado na página 13.
- FASSONE, Riccardo. *Antiquarianism and Hardware Ideology and rhetoric of video game emulation. The case of MAME*. *Tracés* 28 (1), p. 61-80. Paris. 2015. Citado na página 19.
- GEMROT, J.; KADLEC, R.; BÍDA, M.; BURKERT, O.; PÍBIL, R.; HAVLÍČEK, J.; ZEMČÁK, L.; ŠIMLOVIČ, J.; VANSÁ, R.; ŠTOLBA, M.; PLCH, T.; BROM, C. *Pogamut 3 Can Assist Developers in Building AI (Not Only) for Their Videogame Agents*. *Agents for Games and Simulations*. p. 1-15. Springer. New York. 2009. Citado na página 25.

GOMES, Denis dos Santos. *Inteligência Artificial: Conceitos e Aplicações*. Rondônia: FAAR, 2010. Citado na página 12.

GRUNT2084. *Lua scripts to automate Robotron 2084 play on MAME*. 2017. Disponível em: <<https://github.com/grunt2084/robotron-ai>>. Acesso em: 27 mar. 2018. Citado na página 21.

HEDSTROM, Margaret. *Digital Preservation: A Time Bomb for Digital Libraries*. *Computer and the Humanities*, v.31, n.3, Springer. p.189-202. 1997. Disponível em: <https://deepblue.lib.umich.edu/bitstream/handle/2027.42/42573/10579_2004_Article_153071.pdf?sequence=1&isAllowed=y>. Acesso em: 10 nov. 2018. Citado na página 17.

HINGSTON, Philip. *The 2K BotPrize*. In *Computational Intelligence and Games*. CIG 2009. IEEE. Milão. Outubro de 2009. Citado na página 14 e 25.

JOHNSON, M.; HOFMANN, K.; HUTTON, T.; BIGNELL, D. *The Malmo Platform for Artificial Intelligence Experimentation*. In: *Intl. Joint Conference on Artificial Intelligence, New York, 2016*. Disponível em: <<https://www.microsoft.com/en-us/research/wp-content/uploads/2016/07/johnson-malmo-platform-camera-ready.pdf>>. Acesso em: 10 nov. 2018. Citado na página 23.

KONINKLIJKE BIBLIOTHEEK, “*What is emulation?*”, [2017?] data provável. Disponível em <<https://www.kb.nl/en/organisation/research-expertise/research-on-digitisation-and-digital-preservation/emulation/what-is-emulation>>. Acesso em: 10 de abril 2018. Citado na página 17.

LAUREANO, Marcos. *Máquinas Virtuais e Emuladores: Conceitos, Técnicas e Aplicações*. São Paulo: Novatec Editora. 2006. Citado na página 19.

LUA SITE OFICIAL. *A Linguagem de Programação Lua*. 2017. Disponível em: <<https://www.lua.org/about.html>>. Acesso em: 31 mai. 2018. Citado na página 21.

MAGNUN. *Introdução a Bibliotecas em C*. 2012. Disponível em: <<http://mindbending.org/pt/introducao-bibliotecas-em-c>>. Acesso em: 09 jun. 2018. Citado na página 21.

MAMEDEV TEAM. *MAME Documentation Release 0.194*. 2018. Disponível em: <https://docs.mamedev.org/_files/MAME.pdf>. Acesso em: 28 mai. 2018. Citado na página 18.

MAMEDEV.org. *Home of The MAME Project*. 2017. Disponível em: <<http://mamedev.org/>>. Acesso em: 28 mai 2018. Citado na página 14, 19, 20 e 21.

MESS. *Welcome to the MESS Wiki!*. 2017. Disponível em: <<http://mess.redump.net/start>>. Acesso em: 28 mai. 2018. Citado na página 19.

METZ, Cade. *Tech Giants Are Paying Huge Salaries for Scarce A.I. Talent*. 2017. Disponível em <<https://www.nytimes.com/2017/10/22/technology/artificial-intelligence-experts-salaries.html>>. Acesso em: 15 mar. 2018. Citado na página 12.

MURPHY, David. *Hacking Public Memory Understanding the Multiple Arcade*

Machine Emulator. Games and Culture, 8(1):43-53. 2013. Disponível em: <<https://journals.sagepub.com/doi/abs/10.1177/1555412013478687>>. Acesso em: 05 nov. 2018. Citado na página 17.

PEREZ-LIEBANA, D; SAMOTHRAKIS, S; TOGELIUS, J; LUCAS, S. M.; SCHAUL, T. *General Video Game AI: Competition, Challenges, and Opportunities*. 2016. Disponível em: <<https://www.aaai.org/ocs/index.php/AAAI/AAAI16/paper/download/11853/12281>>. Acesso em: 10 dez. 2018. Citado na página 14 e 24.

SCHAUL, T. *A Video Game Description Language for Model-based or Interactive Learning*. IEEE Conference on Computational Intelligence in Games (CIG). 2013. Disponível em: <<https://ieeexplore.ieee.org/document/6633610>>. Acesso em: 10 nov. 2018. Citado na página 23 e 24.

SHANNON, C. *Programming a computer for playing chess*. In: Levy D. (eds) *Computer Chess Compendium*. Springer, New York, NY. 1950. Disponível em: <<https://vision.unipv.it/IA1/aa2009-2010/ProgrammingaComputerforPlayingChess.pdf>>. Acesso em: 05 nov. 2018. Citado na página 13.

Apêndices

APÊNDICE A – Instruções para instalar a biblioteca

Para instalar a biblioteca é necessário realizar os seguintes passos:

1 - Baixar e instalar o Python 3.7.0

<https://www.python.org/downloads/release/python-370/>

2 - Atualizar o pip

```
python -m pip install --upgrade pip
```

3 - Instalar PIL

```
pip install Pillow
```

4 - Instalar win32gui

Baixe a versão desejada do pypiwin32 no link abaixo:

<https://www.lfd.uci.edu/~gohlke/pythonlibs/#pywin32>

E execute o comando:

```
pip install pywin32-224-cp37-cp37m-win32.whl
```

5 - Instalar o *OpenCV*

```
pip install opencv-python
```

Ou

Baixe a versão desejada no link abaixo:

<https://www.lfd.uci.edu/~gohlke/pythonlibs/#opencv>

E execute o comando:

```
pip install opencv_python-3.4.3-cp37-cp37m-win32.whl
```

Ou

Baixe a versão desejada no link abaixo:

<https://pypi.org/project/opencv-python/#files>

E execute o comando:

```
pip install opencv_python-3.4.3.18-cp37-cp37m-win32.whl
```

6 - Instalar a biblioteca *Keyboard*

```
pip install keyboard
```

7 - Editar o arquivo mame.ini

Alterar o valor da propriedade keyboardprovider para dinput

```
keyboardprovider      dinput
```

8 – Baixar os arquivos da biblioteca no link abaixo:

<https://github.com/RicardoDora/MameVirtualPlayer>

APÊNDICE B – Instruções para utilizar a biblioteca

Para utilizar a biblioteca basta importar o arquivo de integração de módulos com o comando abaixo:

```
from mame_lib import *
```

Após importar a biblioteca já será possível acessar todas a funcionalidades da mesma. Exemplos de como utilizar as principais funções:

Obter o quadro atual do jogo que está sendo executado pelo mame e atribuir seu valor à uma variável:

```
img_tela = getImg()
```

Transformar a imagem da tela em uma imagem com apenas um canal de cor, para poder ser utilizada posteriormente pela função simples de detecção de imagem:

```
img_gray = cv2.cvtColor(img_tela, cv2.COLOR_BGR2GRAY)
```

Ler imagem de template no disco rígido e utilizar a função de detecção de objetos para obter as coordenadas onde o template aparece na imagem:

```
img_verde = cv2.imread('imgs2/stank_verde.jpg', 0)
coordenadas = imgMatch(img_gray, img_verde)
```

Destacar com um retângulo verde todas ocorrências do template na imagem original e mostrar na tela:

```
w, h = img_verde.shape[::-1]
for coordenada in coordenadas:
    cv2.rectangle(img_tela, coordenada, (coordenada[0] + w,
    coordenada[1] + h), (0, 0, 255), 2)
cv2.imshow('resultado', img_tela)
cv2.waitKey(0)
```

Ler imagem de template no disco rígido e utilizar a função de detecção de objetos com cor para obter as coordenadas onde o template aparece na imagem:

```
img_tela = getImg()
template = cv2.imread('imgs2/stank_amarelo_up.jpg', 1)
coordenadas = imgMatchComCor(img_tela, template)
```

Ler imagem de template no disco rígido e utilizar a função de detecção de objetos com transparência para obter as coordenadas onde o template aparece na imagem:

```
img_tela = getImg()
img_ryu = cv2.imread('imgs/sf2_ryu_trans.jpg', 1)
coordenadas = imgMatchComTransparencia(img_tela, img_ryu)
```

Enviar sinal de pressionamento da tecla '5' do teclado, responsável pela inserção de moedas no emulador, esperar aproximadamente o tempo de um frame (0,016666 segundos) e soltar a tecla:

```
SendInput(Keyboard(MAME_5))
time.sleep(0.01666)
SendInput(Keyboard(MAME_5, KEYEVENTF_KEYUP))
```