1. A Multi-algorithm Visual Game Playing Agent

Nick Dieffenbacher-Krall

This report describes the design and implementation of a model-based agent for automatically playing part of a role-playing video game. The agent is given realistic constraints: the agent percepts are an image of the game screen, and its actuators are the same (virtual) controls a player would use. The purpose of this project is to experiment with the ways in which multiple AI techniques and algorithms can be combined to produce complex behavior in a single agent, and to compare this work with other approaches such as deep neural networks. The source code for the agent can be found in the `src` directory along with this document.

Table of Contents

2. Introduction

Released in 1989, the Gameboy as the most advanced gaming system of its time. It contained impressive hardware, such as a 4.19 MHz CPU [1]. It also supported impressive games, such as the classic RPG Pokemon Red, released in 1998 [2].

3. Background

The primary goal of this project is to demonstrate the use of multiple AI techniques, which we have discussed over the semester in COS 470 (the secondary goal is training Pokemon!). For this aim, it is necessary to review the class material on relevant artificial agent types, algorithms, and techniques, to decide which will be useful for the project.

3.a. Previous Work

Algorithmic

Neural Networks

Finally, an interesting example of previous work is found within the code of Pokemon Red itself. In the game, a player encounters two types of enemies: wild Pokemon and Trainers. A wild pokemon is a new Pokemon that the player can catch, though they may also wish to fight to wild Pokemon in order to gain experience for their own Pokemon. A trainer's Pokemon can only be fought for experience, or to progress the game. These different types use their own algorithms to determine which move to use against a player. Wild Pokemon simply choose a random move. However, trainer Pokemon use a

4. Program Design

The agent has been implemented using Python 3.11, running on Linux, and using a Wayland (wlroots)-based Desktop environmet. The mgba [3] emulator was chosen to run the actual Pokemon game, using a legally obtained ROM image. This emulator was chosen because it is open source, supports many operating systems, and emphasized low performance requirements. It would not be difficult to port the agent program to other desktop environments or operating systems.

The agent is implemented using a number of open source Python libraries:

Dependency	Purpose	Link
`flake8`	Linting Python code	pycqa.org
`pillow`	Image Processing	pypi.org
`numpy`	The standard Python library for data frames	numpy.org

4.a. Input Layer

The role of the input layer is to capture the current visual output of the game emulator, and translate it into pieces which can be used by the image recognition layer. This layer acts as the percept for the overall game agent.

The agent currently supports running on the Wayland desktop environment. Because Linux Desktop APIs were not a focus of this project, a very simplistic mechanism is used to load screen captures. The `grim` [4] tool is used to save a screen capture of an entire monitor, which is saved to a temporary file. The tool is invoked using the `subprocess` module in Python. The capture file is then read using `Pillow`, and deleted. This is a potentially slow process for gathering an agent percept. In a game-playing agent which required more real time action, such as one that played a 1st person action game, a different input layer architecture would likely be required. One approach would be to read the screen capture directly into the agent program, with no temporary file or external executable. On Wayland/Linux this could be accomplished using the `wlr-screencopy` Wayland protocol [5].

Before it can be processed by the image recognition layer, the game screen capture must be normalized. A Gameboy Color has a screen resolution of 160 x 144 pixels [1]. This means that the screen has a 10:9 aspect ratio, rather than the 4:3 or 16:9 ratios more commonly used today. For this reason, the first processing step is to trim the borders off of the image to create a sub-image containing only game pixels. Next, the input resolution is reduced to more closely match the original game's resolution. It has a higher resolution to start because of the way in which mGBA renders grpahics. The Lanczos algorithm is used to scale the image, as supported in the Pillow library. This scaing algorithm has potentially higher performance costs, but also yields an image that is higher quality, compared to simpled image resolution scaling algorithms such as Bilinear Samplie.

4.b. Image Recognition Layer

The Image Recognition (IR) Layer uses a series of machine learning models and pre-written algorithms to translate the unstructured agent percept image into a structured representation of the current game state, as shown on the screen. The input to this layer is a normalized screen capture of the last game output, as produced by the Input Layer. The output of this layer is a Python `dataclass` containing information about the current state of the game.

4.b.a. Model Training

4.c. Reflex Layer

The purpose of the Reflex Layer is to simplify the implementation of the other layers by making a decision about how to proceed based on the game state. The game state representation is based on the output of the IR layer and some pre-programmed knowledge about the rules of the game. The rules used by this layer fall can be categorized into five groups based on the action that will be taken.

1. Invalid state is detected.
2. The agent is in the middle of an ongoing action.
3. A battle is ocurring and a decision can be reached trivially.
4. A battle has ended.
5. The minimax layer is needed to determine a next move.

In most cases, the minimax layer will not be invoked. Instead, the Reflex layer will call the Output Layer with a list game inputs. The inputs will be based on the current game state.

Action	**Game Inputs**	**Explanation**
End battle	press A 5x	Once one Pokemon has fainted or been caught
Start battle	press A once, wait	To start the battle

4.d. Minimax Layer

What is it

This algorithm layer is not strictly necessary. It would be possible to make a capable game playing algorithm with only a decision tree. This is actually the approach taken by certain enemie AI algorithms (for trainers) within Pokemon Red. However, a decision tree would lead to limitations where the algorithm will not be able to find the same optimal move that a player could. This is because the decidion tree is inherently limited in the number of moves ahead that can be considered.

Heuristic Function

Example Tree

4.e. Output Layer

5. Implementation

5.a. General Development

5.b. Data Collection and Model Training

5.c. Further Improvements

6. Conclusion

6.a. Lessons in Autonomous Agent Design

Bibliography

[1] Nintendo, "Game boy color technical data." https://www.nintendo.co.uk/Support/Game-Boy-Pocket-Color/Product-information/Technical-data/Technical-data-619585.html (accessed: Dec. 5, 2023).

[2] C. Harris, "Pokemon red review." Accessed: Dec. 7, 2023. [Online]. Available: https://www.ign.com/articles/1999/06/24/pokemon-red

[3] "Mgba." https://mgba.io/ (accessed: Dec. 5, 2023).

[4] S. Ser, *Grim*, (2023). Accessed: Dec. 5, 2023. [Online]. Available: https://git.sr.ht/~emersion/grim

[5] S. Ser, "Wlr-screencopy-unstable-v1.xml." https://github.com/swaywm/wlr-protocols/blob/master/unstable/wlr-screencopy-unstable-v1.xml (accessed: Dec. 5, 2023).