Teaching computers to revel in glorious combat

A Capstone Project
by
Rei Armenia and Matthew James Harrison

Submitted to the Faculty of the
Department of Software Technology
in partial fulfillment of the requirements for the degree of

BACHELOR OF SCIENCE
IN
COMPUTER SCIENCE And INNOVATION

Champlain College
2017

Abstract

A brief description (sentence or two) of the problem and why the reader should care. A brief description of what you did. A purpose statement. Include clarifying context as necessary. A brief overview of the approach/method. A brief overview of the results.

Acknowledgements

This section is optional.

Table of Contents

List of Figures

List of Tables

Chapter 1: Introduction (2-3 pages)

1.1 Problem Statement

Can we create a fair and balanced artificial intelligence that plays video games almost like a human? How can we apply artificial intelligence in entertainment?

1.2 Purpose Statement

We shall create an artificial intelligence that uses the same information available to players in order to make decisions. We shall also closely train the Agent to reproduce human-like on a reliable scale.

1.3 Context

Interest in the field of artificial intelligence has been present since nearly the beginning of computing. However, over the past few years interest in AI has been growing. With the recent popularity of machine learning techniques like neural networks, many researchers have made huge advancements in the field. Companies such Google are constantly appearing in mainstream media for their groundbreaking projects. Google's AlphaGo is an amazing example of the work that can be done; AlphaGo is an AI was able to beat the world's best Go player, and it continues to improve its game.

Other groups are contributing to the advancement of AI, although they are not as easily recognizable. A research platform called ViZDoom is one such example that closely relates to our work. ViZDoom is an Open Source platform that allows users to create agents in the videogame Doom, these game agents are doing what any human player would: trying to maximize their score. However, unlike more conventional game AI, they are playing using only the visual buffer. This approach more closely resembles how humans play the game, and was also a large inspiration to this project. Yet, ViZDoom is still primarily a research platform.

Finding cutting-edge AI techniques in the game industry is quite hard. Many developers find these techniques to be both impractical and excessive. These implementations use up sought after resources. Why should they create more work for themselves when many of the problems that we are looking to solve can be reduced to smaller problems with nice well defined algorithms?

With this information, one may think that almost no one uses advanced A.I. in games, and that is primarily the case. However, some companies are implementing these advanced techniques. Instead of using AI to create a diverse experience, they use it to push sales

for specific features. This is the case with Activision's recently acquired patent, which defines a method for manipulating players into engaging with microtransactions.

1.4 Significance of Project

This project's significance comes from the current scarcity of A.I. projects in entertainment. A.I. is one of the quickest growing fields right now, we think that it is important to create a precedent and show both developers and users what these advanced techniques can be used for. If these techniques are continued to be used to take advantage of users, it is possible that A.I. will be "stuck" in this state.

Chapter 2: Literature (3-5 pages)

2.1 Literature Overview

In *Towards Integrated Imitation of Strategic Planning and Motion Modeling in Interactive Computer Games*, Dublin City University researchers Bernard Gorman and Mark Humphrys train use imitation learning techniques to train an artificial agent to collect items in *Quake II*. Specifically, their agent must traverse three-dimensional environments in order to seek said items out without knowing their locations in advance. Furthermore, the researchers set an additional requirement: the agent must "employ human-like motion," which is to say that it must traverse the map in such a way that its movements appear fluid and natural, as if a human were controlling it. The researchers refer to this "imitation of the player's movement" as *motion modeling*, and it represents the main purpose of their experiments. Gorman et al create their training dataset by retrieving state information from demos. Specifically, they use topology learning techniques to define a Markov Decision Process based on all of the player and pickup positions recorded in the demo file. They then employ value iteration using information about the player's status, such as current health and inventory, which is also read from the demo file. Value iteration eventually assigns a utility to each state in the MDP and, ultimately, allows the agent to make intelligent decisions about where to travel next. Gorman et al proceed to discuss their approach in solving the problem of motion modeling. They define a set of *action primitives* by reading all of the input data from the demo file and then disassociating that data from its context. This allows the agent to, after deciding where to navigate, lay out a series of action primitives that will allow it to reach its goal [3].

Gorman et al cite *Learning human-like Movement Behavior for Computer Games* by Christian Thurau and Christian Bauckhage of Bielefeld University, an experimental study which provides a useful behavioral model while also challenging the conventional approach of creating agents based on simple finite state machines. Thurau et al define a hierarchy of behavior types consisting of reactive, tactical, and strategic behaviors [4]. Strategic behaviors are essentially long-term plans for winning the game, such as which objectives to target and in what order. Contrast this with reactive behaviors, which are atomic, immediate decisions, such as moving in a particular direction, aiming, or firing a weapon. Tactical behaviors sit in between reactive and strategic, representing small-scale decisions such as circle-strafing an opponent or choosing to run away from a battle in order to look for a health pack [3]. To implement an agent that utilizes such behaviors, Thurau et al employ topology learning techniques to train a type of neural network called a *neural gas*, using player position data extracted from demo files. The neural gas produces what are referred to as *position prototypes*. This methodology should sound familiar; Gorman et al use many of the contributions detailed in the paper.

In *Autoencoder-augmented Neuroevolution for Visual Doom Playing*, Cornell University researchers Samuel Alvernaz and Julian Togelius explore the use of an autoencoder alongside a neuroevolution technique called "Covariance Matrix Adaptation Evolution Strat-

egy" or CMA-ES. An autoencoder is essentially a process that simplifies and reconstructs images. The researchers implement their autoencoder using Keras, a high-level machine-learning library. They then the autoencoder's output as the input for their neural network, which they train to gather health pickups [1].

In *Learning to Act by Predicting the Future*, Cornell University researchers Alexey Dosovitskiy and Vladlen Koltun use supervised learning techniques to train an agent to play competitive deathmatches in *Doom*. Dosovitskiy and Koltun do so by using two distinctive data streams: one is high-dimensional and consists of "raw visual, auditory, and tactile input," while the other is low-dimensional and provides basic game state information such as "health, ammunition levels, and the number of adversaries overcome" (p. 1). These are referred to as the *sensory* and *measurement* streams, respectively [2].

2.2 Software

Predecessor or current software projects related to your capstone area. May be libraries that you are incorporating in your project, projects that are inspiring your design, a project you are building on, etc...

2.3 Other Sources

Non-academic sources such as white papers, manuals, blogs, etc.

Chapter 3: Methods (5-7 pages)

3.1 Design

Design details of your project. This section can include various descriptions, figures, diagrams, wireframes, etc that are driving your project implementation.

3.2 Environment

We have opted to use our own personal computers running 64-bit Windows 10 with Ubuntu subsystems as both our platform and development environment. There are several reasons to justify this decision. First, we wanted to ensure that we would have the best possible experience when dealing with GPU computation, and NVIDIA simply has a longer and better track record for supporting Windows than it does for Linux. Second, we wanted to avoid the potential pitfall of our platform restricting us to a smaller library of target games. Our two top candidates were *Quake* and *Rivals of Aether* because both of these games support replays, or demos; however, only one of them natively supports Linux. Last, and with regards to using our own computers for both development and testing, we made the decision to abstain from distributed or cloud computing chiefly to save costs, but also as an aesthetic choice based in the desire to see what our computers are capable of.

Despite all of the above justifications for using Windows, it remains to be said that our project cannot live without Linux. For reasons discussed in the next section, we use a Redis server which runs on an Ubuntu subsystem. The solution may not sound elegant, but it was certainly simple to set up.

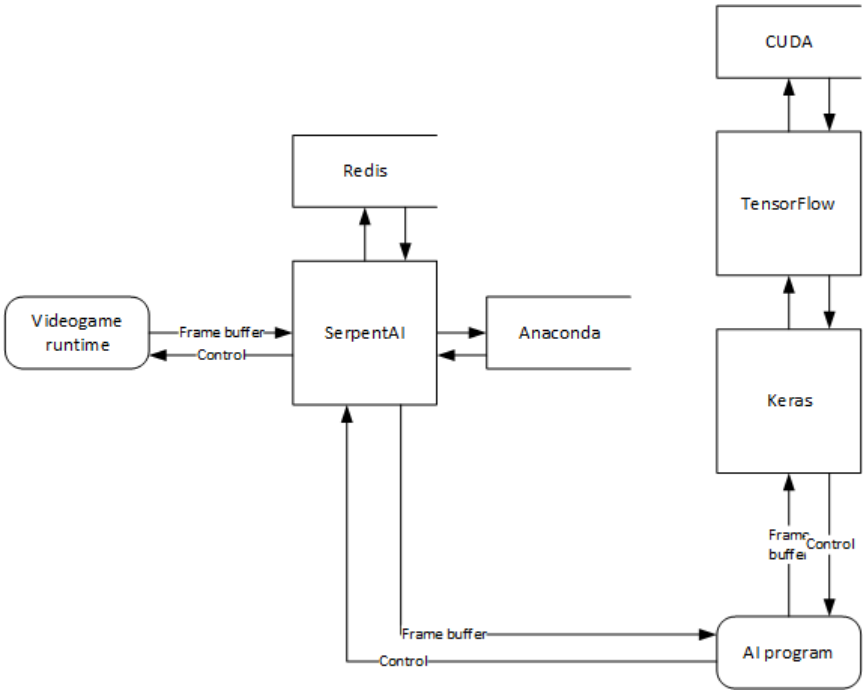
3.3 Frameworks

Our project uses a software stack comprised of the following modules and libraries: SerpentAI, Keras, TensorFlow-GPU, and the Anaconda distribution for Python 3.

We chose to use Python 3 as our primary programming language for its flexibility, intuitive syntax, and data processing capabilities. Python is also helpfully compatible with all of the other software listed above. Furthermore, the Anaconda distribution provides us with a large assortment of Python modules, some of which SerpentAI cites as dependencies. This allows us to more easily work from within a Windows environment, which in turn gives us the most straightforward and stable access to GPU computation via NVIDIA proprietary drivers.

SerpentAI serves as a bridge between our artificial intelligence and its testing environment by providing a simple interface for retrieving frame buffer data from and sending control

Figure 3.1: "Data flow within our software stack"



input to virtually any game runtime. This is achieved by writing a game agent program. Our game agent utilizes TensorFlow, a popular machine-learning library. However, does not access TensorFlow directly. Rather, it uses Keras, a high-level machine-learning library designed to serve as an abstraction layer over foundational libraries like TensorFlow. The low-level library still performs all of the heavy-lifting; Keras simply removes some of the barriers to entry.

3.4 Algorithms

Descriptions and pseudocode of important algorithms in use, relevant, or to be implemented as part of your project.

3.5 Analytical Methods

Descriptions and processes of any analytical methods used as part of your capstone project. This section is most applicable for projects with a data analysis focus or component. Quantitative and qualitative methods.

3.6 Features

A brief overview of primary features you plan to implement.

3.7 Test Plan

A brief overview of what and how you plan to test your project codebase.

3.8 Criteria and Constraints

Project development must meet desired needs within realistic constraints. Many times, constraints are interrelated. Cover the appropriate criteria and constraints related to your capstone. Some example criteria and constraints are health and safety, environmental, political, social, manufacturability, sustainability, economical, and ethical.

Chapter 4: Results (5-7 pages)

4.1 User Interface Design

Design details of your project. This section can include various visualizations (e.g., user interfaces, screenshots) that depict your final project implementation.

4.2 Architecture

Statement and descriptions of systems and structures (e.g., data structures) used in the implementation of your project. This section should detail the inter- and intra-workings of your software and the systems on which the software relies.

4.3 Algorithms

Implementations, explanations, and performance of important algorithms in use as part of your project.

4.4 Analytical Results

Results of quantitative and/or qualitative analysis. This section is most applicable for projects with a data analysis focus or component.

4.5 Features

Describe the planned features which were implemented. Briefly discuss the planned features that were not implemented.

4.6 Testing Results

A brief overview of what and how you tested your project codebase.

Chapter 5: Conclusion (2-3 pages)

5.1 Context

Provide historical, social, cultural, ethical, legal, and other relevant contexts for your project in a reflective manner.

5.2 Challenges and Solutions

Describe particularly important challenges encountered and solutions throughout the development of this capstone project.

5.3 Limitations and Delimitations

Limitations are aspects of a research study that the researcher cannot control. Delimitations are aspects of a research study that the researcher can control.

5.4 Future Work

Where would you like to take the project in the future?

5.5 Project Importance

Why does your project matter? What made this project an important investment of time and resources? Do not reiterate the significance from the Introduction. Rather, be reflective when discussing the project's importance. Discuss what you learned and what your project provides for the computing community.

References

[1] S. Alvernaz and J. Togelius, "Autoencoder-augmented neuroevolution for visual doom playing," *CoRR*, vol. abs/1707.03902, 2017. [Online]. Available: http://arxiv.org/abs/1707.03902

[2] A. Dosovitskiy and V. Koltun, "Learning to act by predicting the future," *CoRR*, vol. abs/1611.01779, 2016. [Online]. Available: http://arxiv.org/abs/1611.01779

[3] B. Gorman and M. Humphrys, "Towards integrated imitation of strategic planning and motion modeling in interactive computer games," *Comput. Entertain.*, vol. 4, no. 4, Oct. 2006. [Online]. Available: http://doi.acm.org/10.1145/1178418.1178432

[4] C. Thurau and C. Bauckhage, "Learning human-like movement behavior for computer games," in *In Proceedings of the Eighth International Conference on the Simulation of Adaptive Behavior (SAB04*, 2004.

Appendix A: Additional Information

As necessary.