Comparing AI Navigation Methods Using Counter-Strike: Global Offensive

Michael Salton
Western University
msalton@uwo.ca

Ethan Pisani
Western University
episani2@uwo.ca

Swayam Sachdeva
Western University
ssachd29@uwo.ca

***Abstract*—This paper describes two AI agents built to play the first-person shooter video game "Counter-Strike: Global Offensive" (CS:GO). The first agent uses a combination of YOLOv7 object detection and A* pathfinding algorithm. The second agent uses a deep neural network trained on a large data set of video footage of professional CS:GO matches using behavioural cloning and offline reinforcement learning. While most "aim-bot" programs tap into the games memory to retrieve information such as the coordinates of players and aim the mouse directly at them, the goal for our project was to create an AI that can match the performance and behaviour of real players while relying purely on the visuals on the screen. We look to compare the behaviour and realism of an AI built with pathfinding vs an AI built with machine learning.**

I. Introduction

Counter-Strike: Global Offensive is a popular first-person shooter video game that has gained immense popularity among gamers worldwide. With the rise of artificial intelligence (AI), there has been increasing interest in developing AI-based systems capable of playing complex games such as CS:GO. In recent years, researchers have focused on developing AI bots that can compete against human players in online matches. These AI bots are designed to learn from the gameplay data, enhance their strategies, and eventually surpass human players in their gameplay performance. However, these projects mostly focused on games with convenient APIs that make it easy to scrap data from the game. Whereas for CS:GO, these tools are not available, which precludes us from using these common techniques.

In this research paper, we present a novel AI-based system that has been developed to play CS:GO. The AI agent is designed to learn and adapt to different game scenarios and player styles. Our research aims to explore the capabilities of machine learning approaches to problems and compare that to the typical pathfinding-like AI, in both video games and real-world scenarios. The paper provides insights into the development of machine learning based systems for gaming applications and the challenges faced in designing effective AI bots for video games and real-world systems.

A. Motivation

The use of behavioural cloning and reinforcement learning presents a promising approach to creating an AI that can effectively play CS:GO. Behavioural cloning, which involves training an agent to mimic the actions of expert players, provides a good starting point for building an intelligent agent. Reinforcement learning, on the other hand, enables the agent to learn from its own experiences in the game environment and improve its decision-making over time.

The ultimate goal of this research is to develop an AI agent that can compete with human players in CS:GO. Such an agent would have numerous practical applications, including enhancing the game experience for players by making AI in video games more realistic and potentially even applying these techniques to real-world scenarios involving AI.

We aim to distinguish between "dumb" and "smart" AI and explore the advantages and disadvantages of each. We define "dumb" AI as relying on traditional pathfinding techniques, where movements are predetermined or precalculated by the computer. This type of AI has been commonly used in video games since the inception of the industry, including popular titles like Mario, Half-Life, Elder Scrolls, and Grand Theft Auto. On the other hand, we define "smart" AI as utilizing dynamic, real-time calculations, that is built with some form of machine learning approach. We will discuss the two AI agents we have built, one "dumb" and one "smart", and see how they compare to each other. What kinds of advantages does either AI bring and what are the problems associated with each technique. Finally, thinking about the future of video games, what kind of AI will be the better option for taking the gaming experience to the next level. Our paper mainly focuses on the navigation aspect of AI. The reason we chose to use CS:GO as our test field as opposed to an environment that is based purely around navigation, like a maze for example, is because; generally in video games, the AI has other actions it performs alongside movement and we wanted to see how these actions affect the AI's movement and what steps need to be taken to keep them from "breaking".

This research will not only contribute to the field of AI by advancing our understanding of how to build intelligent agents that can play complex games like CS:GO, but it will also have practical implications for the gaming industry and beyond. We believe that the results of this research will be of significant interest to game developers, players, and researchers alike, and will pave the way for further research into the interesting overlap of AI and video games.

B. Related Works

We came across a paper on behavioural cloning for CS:GO. Entitled: "Counter-Strike Deathmatch with Large-Scale Behavioural Cloning", the paper goes over how Tim Pearce and Jun Zhu designed a convolutional neural network (CNN) and a long short-term memory (LSTM) neural network to play CS:GO only training on video frames and user input. This is the basis for the "smart" AI model we envisioned. They used 93 hours of online messy data along with a few hours of expert data to train the model. It resulted in a medium-level CS:GO bot in difficulty. The model learned to navigate the 3d environment and was able to identify and shoot enemy players. It also learned human-like strategies like spray control while also having its own strategies. It can still be beaten by a human more times than not, but it is very promising research into the aspect of learning strategy from watching players play the game.

C. Problem Definition

A simple path-finding-like approach may be an easy and common solution for many of today's technologies such as video games, exploration, and industrial automation, to name a few. With video games, probably the most obvious example, pathfinding algorithms have been in place since the beginning. Most games work by having the computer direct characters around the map via a predefined path, or a set of predefined locations. The developers can set up more intricate characteristics to these paths such as a set of rules stating certain paths are preferable over others as some may contain some kind of "risk" or "reward", as well as dealing with dynamic in-game objects. Much of the time these dynamic in-game objects are also under the computer's control, so the program may have to account for a multitude of different entities and move them all in accordance with each other.

Multi-agent path finding (MAPF) is a pathfinding system that has two or more agents navigating the graph. There is a set of common problems with MAPF which are, vertex conflict (two agents attempt to navigate to the same vertex), edge conflict (two agents attempt to use the same edge in the same direction), swap conflict (two agents attempt to use the same edge but in opposite directions (swap positions)), and follow conflict (when one agent attempts to occupy a position at a given time h that was occupied by another agent at time $h-1$). These conflicts can be hard to deal with and have the potential of causing problems within many different applications of a pathfinding system if not properly handled. In the case of the A* algorithm, a solution for these problems is to move all possible agents from one vertex to a free neighbouring vertex one at a time and then initiate the pathfinding process again. This results in a search space of $|V|^K$, where V is the number of vertices on the map and K, is the number of agents, and a branching factor of $\frac{|E|}{|V|}^K$, where E is the total number of possible outgoing edges that an agent can take. These exponential complexities make the A* algorithm intractable for large-scale multi-agent pathfinding problems, as the search space and branching factor quickly become too large to search efficiently.

Not only can these issues have impacts on the game's performance but it makes for an unrealistic and non-immersive gaming experience. AI in many video games today is known to be rather dumb as they don't react in realistic ways to their environment. A famous example of this is Bethesda's "The Elder Scrolls V: Skyrim". The AI in this game is notoriously unrealistic. Non-player characters (NPCs) in Skyrim often walk into one another, randomly stop following you when they're meant to follow you, and overall have quite non-human-like behaviour. Many players describe poor AI as "a reminder it's just a game", which is not what you want when you are trying to get immersed into a new world. Machine learning approaches may offer a better solution for this problem and that is exactly what we set out to learn with this research.

II. Methodology

The design of our two AI agents was done sequentially, starting with the "dumb" one. We gave this agent the name "Atlas". The first set of data was a collection of screenshots from various CS:GO games, personally collected and labelled by our team. We used this data to train our YOLOv7 object detection model. We used OBS to screen record hours of gameplay and then FFmpeg to convert this footage into individual frames, and Roboflow to label them. In total, our dataset was compromised of about 3000 images. We ensured at least 75% of the images contained people, the remaining were blanks. We labelled all the people found in the images with bounding boxes and applied some augmentations to the data such as blurs, shears, and brightness changes to represent some in-game occasions like flash and smoke grenades.

TABLE I
INFORMATION ON THE NUMBER OF IMAGES USED FOR THE YOLOV7 MODEL.

Type	Training	Validation	Testing	Total
Terrorists (T)	1268	178	7	1453
Counter-Terrorists (CT)	1343	191	7	1541
Blanks	137	25	3	165
All	2748	394	17	3159

A. Object Detection

With data collection complete, we used transfer learning to modify the YOLOv7 model. We trained a model with some augmentation then we increased the model size and image resolution input and refined the augmentations. Our best model ends with 87% accuracy at 0.5map. We then converted the model weights to run using NVIDIA Tensor-RT for inference and got the model to shoot at the detected players. We made a function to move the mouse to the middle of each detected bounded box, 2/3 of the way to the top using pynput. With this done Atlas was able to detect, and shoot enemies in the game.

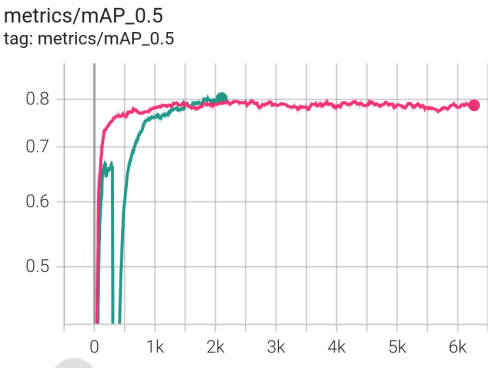

Fig. 1. This map@0.5 graph displays two versions of a the YOLOv7 model, with v1 represented in pink and the v2 represented in teal, where the highest value for v1, 0.78 and 0.81 for v2.

B. Movement

Our second dataset for Atlas was a list of in-game coordinates from around the map representing vertices to be used by the A* algorithm. We went through many different iterations of this graph before we finally landed on one that worked. It was difficult because we had to ensure that the AI could get to any location on the map by strictly walking along the edges between each vertex, as well as ensuring that if two vertices are connected it is possible to walk between them in the game (there are no obstacles obstructing that path). We used this along with a bit of code for sending inputs to the game using the pynput library. The A* algorithm takes a start point and an endpoint on a matrix of 1s representing vertices and 0s representing obstacles with the same cardinality as the waypoint matrix. It then finds the shortest path along the edges from the start point to the endpoint.

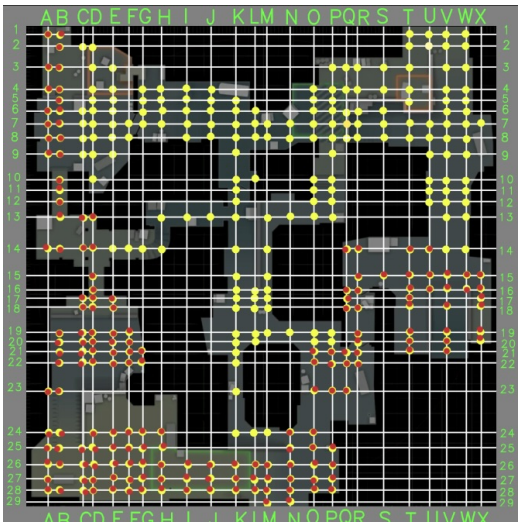

Fig. 2. Graph of vertices used in the A* algorithm.

Unfortunately, the player's current coordinate was not something that was available with CS:GO's game state integration so we were unable to use the in-game coordinates we had collected. Therefore, we had to purely rely on pressing "W", "A", "S", and "D" for a certain length of time, based on the distance between vertices and hope that it would line up perfectly. This caused many issues of its own because if the agent hit any kind of obstacle, like another player, it would be off course and have to be repositioned.

Fig. 3. A* algorithm running on a simulated "Dust II" map.

C. Reinforcement Learning

To optimize the performance of our reinforcement learning agent, named "P-body", we employed a reward function during training. The reward function was defined as follows: $reward = s_8 - 0.5s_{10} - 0.02\sum_{i=1}^{N} s_i$, where s_8 and s_{10} represent specific states and $\sum_{i=1}^{N} s_i$ represents the sum of all states. Our network was configured to accept input dimensions of (200,200,3) for screen pixels, (58) for states, and (62) for output confidence for action space. We utilized behavioural cloning and OpenAI Gym's CS:GO environment to train P-body through offline reinforcement learning. We sourced our training data from past research that involved 93 hours of scraped online data and professional match data containing CS:GO gameplay, presented as a sequence of frames accompanied by player metadata and user input. Offline reinforcement learning was chosen due to its capacity for batch training on powerful hardware, enabling training at a speed that surpasses real-time. JAX/FLAX was used for its exceptional parallelization capabilities and optimized JIT compiler, resulting in improved machine learning training speed. The AI was designed to predict confidence values for user inputs including movement, mouse control, and weapon swapping using the BCLeaner. The incorporation of reinforcement learning allows the AI to learn and improve its performance over time through a reward optimization process, where it receives rewards for

desirable actions and penalties for undesirable ones. Our AI was rewarded for killing enemies during gameplay.

Fig. 4. Graph illustrating the change in loss value for a reinforcement learning model over the course of its training with an end in 58.206.

III. Results

A. Atlas

Atlas would always run into other players which would cause him to be off course from the grid and have to be repositioned. We ran two instances of the AI in the same game, to test multi-agent pathfinding, and the results were like we expected. The two agents would often succumb to vertex conflict, edge conflict, and swap conflict, resulting in them being off-grid and having to be repositioned. Overall, the movement of Atlas seemed very robotic, he often would pause while the next instance of the algorithm is being calculated, and running into obstacles was a common occurrence for him. This type of robotic movement is not something that players want to encounter when playing against or with this type of AI. It makes the game feel fake, reducing players' engagement. Atlas was comparable to the easy built-in CS:GO bots.

B. P-body

On the other hand, P-body was not quite as good at shooting as Atlas was with the YOLOv7 model, but he was much better at moving, and that is really what we are testing. With more training, P-body would be able to become much more proficient at shooting, and eventually overcome Atlas, so that is not an issue. Since P-body did not run using pathfinding the problems with MAPF that Atlas was encountering were not an issue. P-body was smart and was making decisions in real-time, because of this, his movements seemed much more natural compared to Atlas. P-body would never run into another player, obstacle, or wall, he would not make random stops nearly as often as Atlas did, and overall he seemed a lot less robotic and more human-like than Atlas. P-body was comparable to the hard built-in CS:GO bots.

IV. Conclusion

All in all, our project accomplished two AI agents that can traverse an area of space. We set out to determine if pathfinding is an adequate approach to these types of problems or if vying for an alternative approach is worth the time. Atlas the "dumb" AI is compromised of an object detection model used for detecting enemies in the game, and a pathfinding algorithm to move around. P-body the "smart" AI was built using behavioural cloning and offline reinforcement learning, with the aim of cloning the behaviour of professional CS:GO players.

A. "Dumb" vs "Smart"

We determined that using a machine learning approach to AI movement can reduce many conflicts that are present when using a traditional pathfinding approach. Pathfinding has been a common technique in video games and real-world applications for a very long time and perhaps it's time companies start taking advantage of these more modern techniques of machine learning, specifically reinforcement learning. We found that even with the small amount of training our reinforcement learning model had, it was still better at navigating than the pathfinding model. P-body did take more time, knowledge, and resources to construct. It took days of training, data collection and manipulation, programming, and research to be able to get him to function properly. Whereas for Atlas, the amount of knowledge needed to construct him was a lot less. Less time and resources were needed, not to mention there are decades of previous AIs that use the same method. All considered, constructing AI with the "smart" AI approach does produce much higher quality and desirable results.

B. Future

Moving forward, we would like to train P-body even more, and truly see what behavioural cloning is capable of doing. We would like to see him pick up on the "unspoken rules" of CS:GO and see just how close he can mimic the behaviour of humans. The ability to pick up on unspoken rules in video games is particularly important for realistic and immersive gaming experiences. In games like CS:GO, there are often unwritten rules and conventions that players follow, such as how to move around the map, when to engage in combat, and how to coordinate with teammates. If an AI could pick up on these rules and replicate them, it could make the game feel more natural and lifelike. The future of gaming depends on the research being done today and AI is at the front of it. Realistic AI in video games has been something companies have been trying to nail for years, perhaps a machine learning approach will become the mainstream of future video game AI.

References

[1] I. Kostrikov, "JAXRL: Implementations of Reinforcement Learning algorithms in JAX," GitHub, Oct. 2022. [Online]. Available: https://github.com/ikostrikov/jaxrl2. [Accessed: Mar. 13, 2023].

[2] T. Pearce and J. Zhu, "Counter-Strike Deathmatch with Large-Scale Behavioural Cloning," arXiv:2104.04258, 2021.

[3] Pouke, M.: Using GPS data to control an agent in a realistic 3D environment, pp. 87–92. IEEE, September 2013. https://doi.org/10.1109/NGMAST.2013.24

[4] C.-Y. Wang, A. Bochkovskiy, and H.-Y. M. Liao, "YOLOv7: Trainable bag-of-freebies sets new state-of-the-art for real-time object detectors," arXiv preprint arXiv:2207.02696, 2022.

[5] C.-Y. Wang, H.-Y. M. Liao, and I.-H. Yeh, "Designing Network Design Strategies Through Gradient Path Analysis," arXiv preprint arXiv:2211.04800, 2022.

Our GitHub Repository: https://github.com/michaelsalton/ProjectLambda