Reinforcement Learning Within Video Game Development

Illa Rochez

Virginia Tech

illar@vt.edu

Abstract

This project focuses on the testing of Machine Learning principles, primarily in the context of video game development. The project will delve into a few different applications, primarily focusing on a pathfinder utilizing the Unity AI library.

1. Introduction

1.1 Background

One large facet of today's digital age is videogames. A form of entertainment that has taken the world by storm. We've gone from low fidelity graphics to some of the most advanced environments ever seen to date in a span of merely ten years (3). The technological revolution is expanding and developing at a rate faster than any other revolution in human history. However, one area has yet to fully grow until more recently: intractability and adaptability. Over the years the structure of video games has been the same: a developer attempts to predict all the possible ways a player may interact with a system, and then a programmer will hardcode all those inputs to a set list of outputs. This severely limits the way players can interact within the world of a game. Imagine a video game where you could ask non-player characters (NPCs) anything, where you could talk for hours with the shop owner asking for all sorts of items that might not exist in the game and getting suggestions for something similar, NPCs move and react similar to how a player would, avoiding traps and navigating the terrain of the game. This is the type of progress we can look forward to as video games begin to implement machine learning.

1.2 Motivation and Project Description

This project was motivated by an interest in exploring some of the new technologies that are influencing videogames. The goal of this project is to develop a simple path finder agent. A Non-Player Character (NPC) that is controlled by a reinforcement algorithm, will attempt to navigate to a point on a "game level". The point will be determined by a human user upon running the program, and the primary point of interest is that the agent has never "seen" the level it will be attempting to navigate before. The agent will attempt to find the quickest path to the selected point without colliding with the edges of the "track". For this implementation, a Machine Learning Agent plug-in for a popular game development software, Unity, will be utilized to develop the NPC, the test and training environments, and to train the NPC. This project will also take into consideration how this plug-in can improve accessibility to machine learning agents to the general public, many of whom, have little or no knowledge of how to implement a machine learning agent.

1.3 Information about the software used

The library was developed utilizing TensorFlow and it offers a plethora of options for experimenting with Machine Learning, in particular it facilitates experimentation with Reinforcement Learning and Artificial Intelligence [10]. Many of these various features were experimented with during the research phase of this project. The NavMesh features of the Unity software will be the primary aspect of the library that this project will develop a fully functional implementation for. This library features AI agents that can be modified, retrained, and overall manipulated to develop interactive and interesting gameplay.

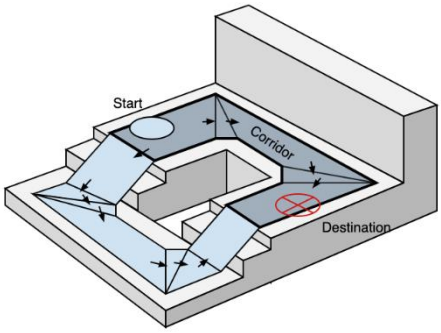

Figure 1.3.1: NavMesh Explanation Diagram

There are three main aspects of the NavMesh implementation that are fundamental to creating a pathfinder unit: the NavMeshAgent, the NavMeshModifier, and the NavMeshSurface [11]. Each is critical to the operation of the pathfinder. The NavMeshAgent has its own library of functions that allow it to take in data and utilize information from the NavMeshSurface to compute the best route and then move towards the selected destination if it is reachable.

The NavMeshModifier interacts with the NavMeshSurface as well. The purpose of this component is to modify how different items within the environment are recognized by the NavMeshSurface. This class can be used to create obstacles like walls, or to create spaces that only certain types of agents can pass through. For example, if you had a human NPC and a mouse NPC navigating some sort of terrain. If you build the environment to be realistic, there are some areas that a mouse could reach that a human could not, and vice versa. The NavMeshSurface enables these sorts of distinctions between when an object is walkable and to what agent types the object is considered to be walkable.

The last type to consider is the NavMeshSurface. This is the walkable space that is generated via a "bake" method. Once this walkable surface is created, it is stored as a collection of convex polygons. The NavMesh uses the stored polygons to calculate the best route to a selected destination location from the starting point. This process can be seen upon running the project. It determines the shortest path by collecting sequences of polygons that lead from the starting point to the destination point and performing computations utilizing the A* algorithm to determine the shortest path.[15].

The A* algorithm has many similarities to Reinforcement Learning which was the intended method of implementation for this project. Back propagation is utilized within the A* algorithm to determine the best route, similar to how Neural Networks developed for Reinforcement Learning would function for the same problem. For more information about the A* algorithm see reference 15 and reference 14.

The reasoning behind the choice of an algorithm like A* over pure Reinforcement Learning within pathfinding would be the concept of exploration versus exploitation. "Reinforcement Learning is somewhat like the heuristic function in A*, except that it's updated as the agents try new things and learn what works" [14]. In game development it is more common to utilize the A* algorithm as a basis for a pathfinding agent, and then to utilize Reinforcement Learning to guide to teach the agent how to behave in the world. After learning about the A* function that the Unity AI library utilizes, it was determined that it would be a more efficient and more realistic implementation of a pathfinder, similar to what is used in the game development industry.

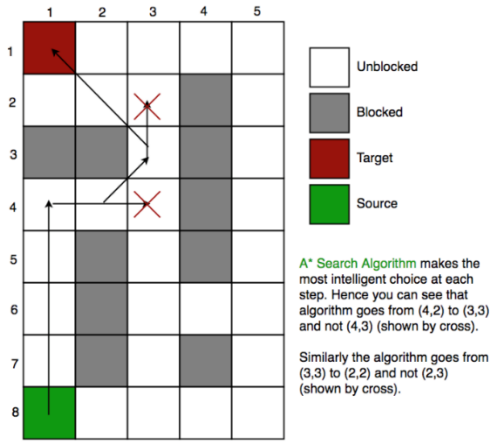

Figure 1.3.2: A* Search Algorithm Explanation

1.4 Current Implementation

Today, a pathfinder agent is a very common type of learning agent found in a plethora of videogames, especially First-Person Shooter games. Having an NPC that can navigate an area, avoid obstacles, and interact with its environment without using a manually programmed script allows for more innovative and creative interactions between player characters and NPCs. Instead of an NPC having a set path of movement, it could continue to move in unpredictable and a more human-like fashion, leading to a more enjoyable and realistic experience.

Within the current practice, there are few limits to what could be achieved with reinforcement learning theoretically. The only real limits theoretically are the imagination. In terms of practicality, however, there are many cases where a system simply doesn't have the computational power needed to support such complex algorithms. The main goal of the community is to make these concepts a reality either through improving our computers specs or finding ways to reduce the complexity to a level that today's technology can support. Many companies use videogame development to support research into the many different applications of reinforcement learning, in addition to growing the complexity and capabilities of pre-existing reinforcement learning agents.

1.5 Results of Successful implementation

In the case that this project is successful, it will serve as a proof of concept for the usability of Machine Learning in videogame development. Additionally, the success of this project will illustrate a growing accessibility to machine learning. The success of this still experimental plug-in will result in a greater focus on the creation of compelling and creative game play instead of limiting the imaginative process of a game developer by their technical understanding of machine learning agents. The plug-in also aims to educate with a large library of agents to study, reprogram, and experiment with; allowing users the chance to grow their understanding of machine learning through experience.

2. Approach

2.1 Initial Research and Findings

Before finalizing the design of the project, a variety of different machine learning tools for game development were explored and considered. One of note would be the Google semantic reactor [6]. This software allows the user to compare an inputted phrase to a potential output based on the semantic relevance of the words used in the input to the words in each of the possible outputs. A small amount of time was spent investigating this software and experimenting with the potential implementations. Though this software was not used for this project, experimenting with some of the features proved helpful in developing an understanding of the types of machine learning technology currently available.

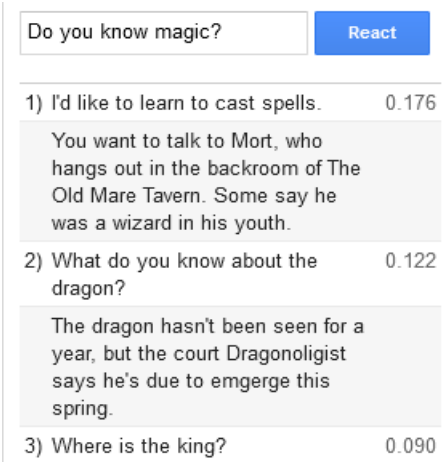

Figure 2.1.1: Example output from the Google Semantic Reactor

After some consideration, a decision was made to have the focus of the project be implementations of Machine Learning within the Unity Game Development Engine.

2.2 Project Design and Process

2.2.1 Establishing Project Concept.

The project began with a very broad concept: create a pathfinding agent within the Unity Game Design Engine. The first step was to refine this concept into a more concrete understanding of what would need to be accomplished to complete this goal. This project was inspired by a project conducted by a youtuber named Jabrill [5], however, there was no distinct step process outlined within the content provided. The videos were more theory and education focused, with no source files to look through. Using the concept, he outlined; however, more research was able to be conducted about how to implement a pathfinder agent, which led to the discovery of the Unity AI library.

The initial design for this project entailed researching the Unity Machine Learning Agents and how to utilize them to achieve the desired results. In addition to learning more about the plug-in, it was also necessary to establish a baseline foundation in C#, the coding language utilized by the Unity game development engine. This research led to a tutorial developed by Tim Bonzon, [8] was very insightful. The tutorial focused on explaining how to create a script for training an agent within unity, including all of the libraries that could be downloaded and a step-by-step guide on how to modify the script for training a ball balancing agent. The most fascinating part of this process was being able to see the training process, in addition to the final results. The tutorial was a great introduction to the different types of agents offered by the Unity AI library, as well as some of the features of game play that could be implemented with these agents.

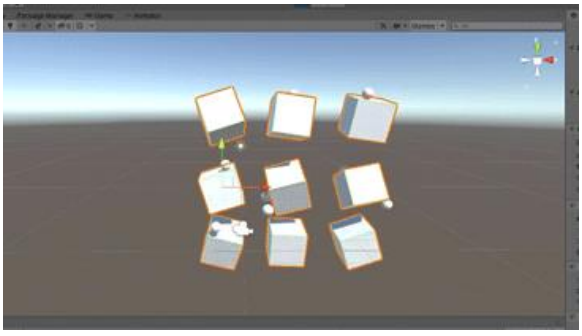

Figure 2.2.1.1: GIF of Ball balancing agents from the ball balancing tutorial

After experimenting with the ball balancing agent, the next phase was to select a feature within the Unity AI library to implement a pathfinder agent. Within the

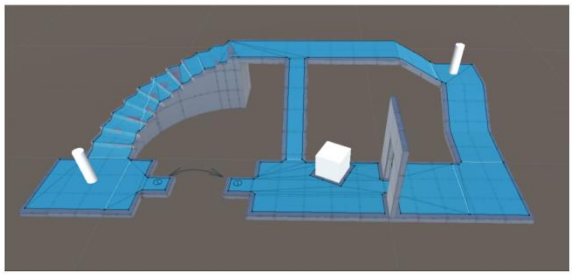

Figure 2.2.1.2: An example of Unity NavMesh

Unity AI library, there were several different routes available to choose from for implementing some form of machine learning element within the context of a videogame, however a NavMesh was the one selected for this project. A NavMesh is a collection of convex polygons that covers all walkable surfaces within a region [12]. The next step was to then determine how to use the NavMesh features to create a pathfinder.

While researching the NavMesh, a set of tutorials for the NavMesh was found on Unity's tutorial site. This particular tutorial utilized content created by a youtuber with the screen name "Brackeys" as further reference for developing a NavMesh implementation The information contained within the tutorial and the base implementation code was used to develop this project. (See section 6 to access the GitHub)

2.2.2 Implementation of Concept

This project was implemented by following and modifying the tutorial provided by Unity Learn [13]. The first step was to download the necessary Unity Libraries and set up the programming environment in addition to the Unity Interface Environment. Unity utilizes Visual Studio as a code editor. It was necessary to ensure that packages for editing code written in the C# language were installed as well.

After this basic environment was set up, the process of creating the unity project for the implementation was started by following the tutorial and making small changes along the way. This process began with adding NavMesh components to basic Unity objects and modifying the pre-programmed features of each to suit the needs of the project. Once this process was completed, a simple pathfinder agent script was implemented using C#. The script checks for user interaction with the environment, and if an interaction is detected, the program will interact with the NavMeshAgent. User interaction is defined as clicking on a valid region of the game space. This selected location is passed to the NavMeshAgent as a destination to navigate towards.

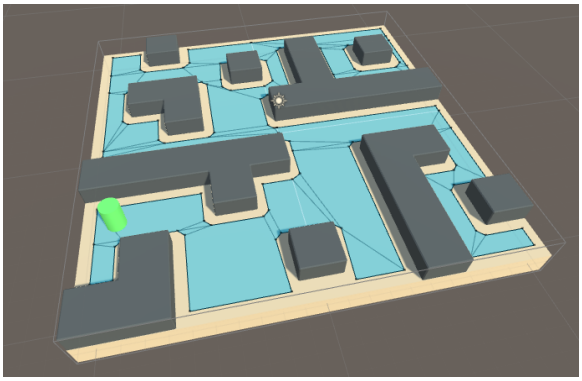

Figure 2.2.1.3: Pathfinder Implementation with NavMesh area visible.

2.3 Anticipated and Encountered Problems

One of the largest anticipated problems was the lack of experience working with Unity. The ability to problem solves and troubleshoot code errors will be heavily limited by having a foundation in the C# language and having a basic understanding of the Unity development environment. Another anticipated problem would be the creation of the training and testing spaces. A large number of "tracks" will most likely be necessary to achieve verbose and detailed training, so it is highly possible that the success of the project may be limited on how much training the agent will be able to do in various unique environments.

During the implementation of this project there were many issues that were encountered. The biggest issue was the initial environment setup. The version of Unity that the Brackeys tutorial uses is a few years old and using the newer version of Unity created additional problems that were resolved once an older version of Unity was installed. There were also a lot of issues with getting Visual Studio to recognize C# code. While these specific issues weren't expected, some level of technical issues with the environment were predicted. Being unfamiliar with the software used on any project can lead to bottlenecks in the development process.

Another difficulty was encountered however that was not expected. After finishing the initial pathfinder tutorial and hitting run, the pathfinder didn't function as expected. If the terrain was selected, the agent wouldn't move, however the agent would shift slightly if it was selected. With a limited amount of experience utilizing Unity, the problem was swiftly resolved. Unity utilizes a feature called a "collider" to allow for object interaction. It's how you could launch one box into another and see them interact similar to how two boxes thrown together in reality would interact. "Colliders" also allow for a user to interact with an

object. The program was unable to recognize the user clicking on the ground because there wasn't a collider on the ground object. After adding a collider, the program functioned as expected and was fun to play with, watching the agent maneuver around the space.

3. Experiments and Results

The experimentation and results of this project are much different from a project that would explore a theoretical solution to a problem. Instead, this project focused on the exploration of existing technologies and proving the concepts they claimed. Multiple different forms of a machine learning implementation within the sphere of video game development were encountered, including two brief explorations of Semantic Machine Learning and Reinforcement Learning. A heavy focus was placed on the implementation of a pathfinder agent, however, during that process, other agent types were discovered and considered.

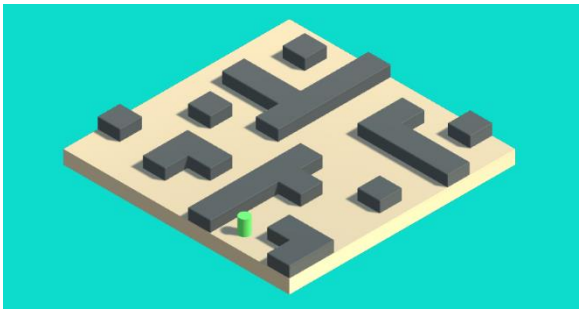

Figure 3.1: Actual Gameplay space developed

Overall, the results of this project are considered to be successful. A pathfinding agent was successfully implemented within the Unity Engine, and it will be modified and played with to further understand the technologies being utilized within the software. Experimentation with the various code libraries shows that a user-friendly approach to machine learning implementations is freely available and capable of creating a copious number of unique projects.

4. Availability and Reproducibility

This project will be easily reproducible by others. This is primarily due to the nature of the Unity Machine Learning Agents Plug-in and the various tutorials referenced. They were developed for beginners, as a way to enter the sphere of machine learning implementations within game development. Additionally, with the usability of the Unity interface, this kind of agent will be able to be reproduced by anyone with a minimal understanding of machine learning and its applications.

The Machine Learning AI GitHub as well as a GitHub for the pathfinder project are open source and free to access. The code and testing environments developed for this project will also be freely accessible via GitHub, allowing others easy access to the modified implementation of the original project in addition to the unmodified tutorial files provided by the Unity Learning site. The results of this project will be easily reproducible by anyone with Unity installed (ver. 2017.3) and the project files downloaded. (See section 6 to access the GitHub Repositories)

5. References

[1] Alonso, Eloi, et al. "Deep Reinforcement Learning for Navigation in AAA Video Games." ArXiv.org, 17 Nov. 2020, https://arxiv.org/abs/2011.04764.

[2] authors, All, and Matthew E. Taylor. "Reinforcement Learning Agents Providing Advice in Complex Video Games." Taylor & Francis, https://www.tandfonline.com/doi/full/10.1080/09540091.2014.885279.

[3] Bowling, Michael, et al. "Machine learning and games." Machine learning 63.3 (2006): 211-215 "Building Smarter Games with Machine Learning - Youtube." Youtube, Google Cloud Tech, https://www.youtube.com/watch?v=30y9zk5COqw.

[4] Markowitz, Dale. "Build Apps Powered by Language with Semantic ML." Dale on AI, 5 Aug. 2020, https://daleonai.com/semantic-ml.

[5] Reinforcement Learning: Crash Course AI#9 - Youtube. https://www.youtube.com/watch?v=nIgIv4IfJ6s.

[6] "Semantic Reactor - Semantic Experiences." Google, Google, https://research.google.com/semanticexperiences/semantic-reactor.html.

[7] "Stickman A.I. Learns to Walk." Stickman A.I. Learns To Walk | Unity Copilot - BETA, https://unitycopilot.com/videos/view/kowCrRPqRMM.

[8] Technologies, Unity. "Machine Learning Agents." Unity, https://unity.com/products/machine-learning-agents.

[9] Bonzon, Tim. "An Introduction to Unity's ML-Agents." *GameDev Academy*, 15 Apr. 2022, https://gamedevacademy.org/unity-ml-agents-tutorial/.

[10] Technologies, Unity. "AI." *Unity*, https://docs.unity3d.com/Manual/com.unity.modules.ai.html.

[11] Technologies, Unity. "Building a Navmesh." *Unity*, https://docs.unity3d.com/Manual/nav-BuildingNavMesh.html.

[12] Technologies, Unity. "Inner Workings of the Navigation System." *Unity*, https://docs.unity3d.com/Manual/nav-InnerWorkings.html.

[13] "Unity Navmesh." *Unity Learn*, https://learn.unity.com/tutorial/unity-

navmesh?projectId=5f60d859edbc2a001ee947ea#5c7f8528edbc2a002053b497.
[14] *Ai Techniques*, http://theory.stanford.edu/~amitp/GameProgramming/AITechniques.html#:~:text=Pathfinding%20is%20often%20associated%20with,were%20developed%20by%20AI%20researchers.
[15] "A* Search Algorithm." *GeeksforGeeks*, 13 Apr. 2022, https://www.geeksforgeeks.org/a-search-algorithm/.

6. GitHub

Unity Learn Tutorial Files:

https://github.com/Brackeys/NavMesh-Tutorial

Unity AI Library:

https://github.com/Unity-Technologies/ml-agents

Project Source Files:

https://github.com/illar47/ECE4424FinalProjectCode.git