DEVELOPMENT OF A 2D PLATFORMER GAME AND MACHINE LEARNING MODEL

by

MATHEW MICHAEL DAWSON
URN: 6743842

A dissertation submitted in partial fulfilment of the requirements for the award of

BACHELOR OF SCIENCE IN COMPUTER SCIENCE

May 2025

Department of Computer Science
University of Surrey
Guildford GU2 7XH

Supervised by: Daniel Gardham

I declare that this dissertation is my own work and that the work of others is acknowledged and indicated by explicit references.

Mathew Michael Dawson
May 2025

Abstract

This dissertation explores deep reinforcement learning in video games by developing a Deep Q-Network (DQN) agent capable of autonomously playing a custom 2D platformer game. The game is built using the Godot, a lightweight open-source game engine, and the model is implemented using the PyTorch deep learning framework. The model is lightweight enough to run on the CPU, but can be accelerated using a GPU. Experimental results show the DQN agent successfully learned effective strategies, achieving times comparable to human players. The agent's performance improved with training, demonstrating the potential of deep reinforcement learning in video games.

Acknowledgements

I would like to thank my supervisor Dr Daniel Gardham for being on board with my idea from our first meeting, as well as his continued guidance and support, had it not been for him I would likely have picked something much more straightforward and within my comfort zone. I would also like to thank my friends and family for encouraging me along the development of this project. Thanks also go to the Godot contributors who work tirelessly, for free, on this wonderful game engine.

Contents

List of Figures

List of Tables

Abbreviations

AI	Artificial Intelligence
DQN	Deep Q Network
ML	Machine Learning
RL	Reinforcement Learning
DRL	Deep Reinforcement Learning
MDP	Markov Decision Process
NPC	Non-Player Character
FSM	Finite State Machine
BT	Behaviour Tree
GOAP	Goal-Oriented Action Planning
CPU	Central Processing Unit
GPU	Graphics Processing Unit

Chapter 1

Introduction

1.1 Chapter Overview

This chapter will focus on introducing the project with an overview along with its objectives and limitations.

1.2 Project Background

In the area of video games, Artificial Intelligence (AI) has been used for many years to create non-player characters (NPCs) that can interact with players in a believable way. This was first seen in the game "Nim" in 1948 (Wikipedia 2025). This is often done using finite state machines (FSMs) or behaviour trees, which allow NPCs to react to player actions in a way that seems intelligent (Carpenter 2019). However, these methods can be limited in their ability to adapt to new situations, due to being based on pre-defined rules and behaviours. For example, developing an FSM for a procedurally generated game, or one with another amount of randomness involved, can be difficult or even impossible, as the FSM must be able to handle all possible situations that may arise. This is the problem that I will be experimenting with and attempting to address in this project. Reinforcement Learning (RL) provides an alternative approach to video game AI. Unlike traditional methods, RL agents learn through interaction with their environment by receiving rewards for desirable actions. This allows them to develop adaptive strategies without explicitly programmed rules. Building upon RL, Deep Reinforcement Learning (DRL) combines traditional RL algorithms with deep neural networks, enabling agents to process complex visual inputs and learn effective policies from high-dimensional data. DRL has demonstrated remarkable capabilities in video games, as seen in systems like OpenAI's DQN that mastered Atari games and AlphaGo which defeated world champions in Go. The adaptive nature of DRL makes it particularly promising for procedurally generated or dynamic game environments where traditional AI approaches struggle.

1.3 Project Overview

This project will begin with research and literature review into traditional approaches to AI in video games, as well as reinforcement learning and deep reinforcement learning techniques. Following this research phase, a simple game environment will be developed using the Godot game engine, designed specifically to test and showcase the capabilities of an RL agent. The project will then implement and train an RL agent to operate within this environment, focusing on creating NPCs that can learn and adapt to changes within the game rather than following predetermined patterns. The performance of these agents will be evaluated against traditional AI methods, and the final implementation will include an RL agent as some part of the game.

1.4 Project Aim and Objectives

The primary aim of this project be to attempt a non-standard machine learning based apporach to video game AI.

- Research and understand how existing AI in video games works, then the fundamentals of Reinforcement Learning (RL) and its application in video games.
- Design and implement a video game environment suitable for testing RL agents.
- Develop and train an RL agent to interact with the game.
- Evaluate the performance of the RL agent.
- Implement the RL agent as part of the game, and have it interact with players.

1.5 Limitations

This project will have the following limitations:

- The game will be custom made, rather than an existing one.
- The game will have simple graphics and gameplay, acting more as a "front-end" for the model which will be the main focus of the project.
- The model will be lightweight, and will need to run on a single CPU and/or GPU. It must also not consume too much memory. This is to ensure that I can train and run it on my hardware.

Chapter 2

Literature Review

This chapter contains a review of traditional approaches to AI in video games. It then explores RL and deep learning fundamentals, algorithms and data structures, as well as some of their existing applications to video games.

2.1 Traditional approaches to AI in video games

This section explores some existing traditional methods of implementing AI in video games.

2.1.1 Finite State Machines

A Finite State Machine (FSM) is a computational model that has been foundational in video game AI development for decades. FSMs represent an agent's behavior as a set of discrete states, with well-defined transitions between these states triggered by specific conditions or events (Spiceworks 2021). Each state encapsulates a particular behavior or action pattern, while transitions define the rules governing when an agent should change its behavior. The simplicity and predictability of FSMs make them particularly suitable for controlling NPCs with straightforward behavioral patterns, as they are computationally efficient and easily debuggable. However, FSMs face significant limitations as complexity increases: the number of states and transitions can grow exponentially, leading to the "state explosion" problem that makes maintenance challenging. Additionally, FSMs struggle with handling concurrent behaviors and can appear rigid when compared to more dynamic AI approaches. Despite these limitations, FSMs remain prevalent in game development due to their intuitive implementation and reliable performance for many common AI tasks.

2.1.2 Behaviour Trees

Behaviour Trees (BTs) represent a significant advancement over FSMs in game AI architecture, offering a hierarchical, modular approach to decision-making. Originally developed for robotics and adopted by the game industry in titles like Halo 2 (Carpenter 2019), BTs organize agent behaviors into a tree structure where leaf nodes represent atomic actions and internal nodes control flow through various composites such as sequences, selectors, and parallels. This structure enables developers to create complex, reusable behavior patterns that can be visually represented and intuitively understood. Unlike FSMs, BTs naturally handle concurrent actions and gracefully manage behavior prioritization through their hierarchical evaluation. BTs excel at creating responsive AI that can react to changing game conditions while maintaining coherent behavior patterns. They facilitate an incremental development approach, allowing designers to progressively refine AI by adding branches without disrupting existing functionality. While BTs require more initial design consideration than FSMs, their scalability, maintainability, and ability to represent sophisticated decision-making logic have made them the standard approach for contemporary game AI systems, particularly in action, strategy, and open-world games where adaptable NPC behavior is critical to player experience.

2.1.3 Goal-Oriented Action Planning

Goal-Oriented Action Planning (GOAP) represents a more dynamic approach to AI decision-making compared to FSMs and BTs. GOAP employs principles from automated planning and means-end analysis to create AI agents that formulate plans to achieve specific goals. Unlike more rigid systems, GOAP agents dynamically determine action sequences by considering current world states, available actions with preconditions and effects, and desired goal states. In a GOAP system, each action is associated with both preconditions (requirements that must be satisfied before the action can be taken) and effects (how the action changes the world state). The AI agent uses planning algorithms, commonly A* search, to find the optimal sequence of actions that transforms the current state into the goal state. This approach allows NPCs to solve problems creatively and adapt to unexpected changes in the game environment. GOAP gained prominence through its implementation in F.E.A.R. (2005) (Thompson 2020), where it produced enemies capable of contextually appropriate tactical behaviors like seeking cover, flanking the player, and coordinating with allies. The system's strength lies in its separation of goals (what the agent wants to achieve) from the specific methods to achieve them, creating emergent behavior that can surprise even the developers. While GOAP offers exceptional adaptability and problem-solving capabilities, it comes with higher computational costs and increased implementation complexity compared to FSMs and BTs. Despite these challenges, GOAP remains valuable for games requiring sophisticated AI that can respond intelligently to dynamic and unpredictable gameplay scenarios.

2.2 Reinforcement Learning

This section explores fundamental RL concepts and algorithms.

2.2.1 RL Fundamentals and the Markov Decision Process

Reinforcement Learning (RL) represents a departure from traditional game AI approaches by focusing on learning optimal behaviors through trial and error interaction with an environment. Unlike FSMs, BTs, or GOAP systems that rely on pre-programmed rules, RL agents improve their decision-making capabilities through experience.

At its core, RL is formalized as a Markov Decision Process (MDP) consisting of:

- A set of states S representing all possible situations an agent may encounter
- A set of actions A that the agent can take
- Transition probabilities $P(s'|s, a)$ defining the likelihood of moving to state s' after taking action a in state s
- A reward function $R(s, a, s')$ providing feedback on the quality of decisions
- A discount factor $\gamma \in [0, 1]$ determining the importance of future rewards

The agent's objective is to learn a policy π that maps states to actions in a way that maximizes the expected cumulative reward. This optimization process balances immediate rewards against long-term consequences, addressing the fundamental exploration-exploitation dilemma: whether to capitalize on known good strategies or explore new possibilities that might yield better results.

RL algorithms generally fall into three categories: value-based methods (like Q-learning), policy-based methods (such as policy gradients), and actor-critic approaches that combine aspects of both. The choice of algorithm depends on factors including the complexity of the state space, whether the environment is fully observable, and computational constraints. Unlike traditional AI techniques, RL offers adaptability to unexpected situations and can discover novel strategies beyond designer expectations. However, these advantages come with challenges including high sample complexity (requiring many environment interactions), difficulty in specifying appropriate reward functions, and potential convergence to suboptimal solutions when trained in limited scenarios.

2.2.2 Deep Reinforcement Learning

Deep Reinforcement Learning (DRL) combines classical RL with deep neural networks to handle high-dimensional state spaces that would be intractable with traditional tabular methods. This integration enables RL to operate effectively in complex environments with visual inputs, continuous action spaces, and intricate state representations common in modern video games.

The breakthrough Deep Q-Network (DQN) algorithm, introduced by DeepMind in 2015, demonstrated superhuman performance on Atari games using only pixel inputs and game scores. DQN employs several key innovations including:

- Experience replay, which stores and randomly samples past experiences to break correlation between sequential samples
- Target networks that stabilize training by reducing moving target problems
- Convolutional neural networks that process visual information effectively

Chapter 3

Bibliography

Carpenter, S. (2019), 'Simulacrum: Building believable behaviors in video game ai', https://medium.com/@selcarpe/https-medium-com-simulacrum-28477a0e759e. Accessed: 26/04/25.

Spiceworks (2021), 'What is finite state machine (fsm)?', https://www.spiceworks.com/tech/tech-general/articles/what-is-fsm/. Accessed: 26/04/25.

Thompson, T. (2020), 'Building the ai of f.e.a.r. with goal-oriented action planning', https://www.gamedeveloper.com/design/building-the-ai-of-f-e-a-r-with-goal-oriented-action-planning. Accessed: 26/04/25.

Wikipedia (2025), 'Artificial intelligence in video games', https://en.wikipedia.org/wiki/Artificial_intelligence_in_video_games. Accessed: 26/04/25.