# UNIVERSITY OF LONDON
## INTERNATIONAL PROGRAMMES

## BSc Computer Science and Related Subjects

# CM3070 FINAL PROJECT

FINAL REPORT

Kane and Abel: AIs that play games

Total words: 9445

# Table of Contents


**CHAPTER 1: INTRODUCTION ..... 5**

**1.1 ABSTRACT ..... 5**
**1.2 PROJECT TEMPLATE AND IDEA ..... 5**
**1.3 MOTIVATION ..... 6**
**1.4 RESEARCH QUESTION AND AIM ..... 6**
**1.5 DELIVERABLES ..... 7**
**1.6 DOMAIN AND USERS ..... 7**

**CHAPTER 2: LITERATURE REVIEW ..... 8**

**2.1 INTRODUCTION ..... 8**
**2.2 BACKGROUND ON FINITE STATE MACHINES AND REINFORCEMENT LEARNING ..... 9**
2.2.1 FINITE STATE MACHINES ..... 9
2.2.2 REINFORCEMENT LEARNING ..... 10
2.2.3 SIGNIFICANCE IN ROBOTICS AND AI ..... 10
**2.3 REVIEW OF LITERATURE ..... 11**
2.3.1 FSM AND RL IN QUADRUPEDAL LOCOMOTION ..... 11
2.3.2 FSM AND RL IN AUTONOMOUS DRIVING ..... 11
2.3.3 FSM AND RL IN HUMANOID ROBOT ..... 12
2.3.4 FSM AND RL IN TESTING OPTIMIZATION ..... 13
**2.4 COMPARATIVE ANALYSIS OF FSM AND RL ..... 14**
2.4.1 STRENGTHS AND WEAKNESSES OF FSM AND RL ..... 14
2.4.2 HYBRID APPROACHES ..... 14
2.4.3 INDIVIDUAL USE OF FSM AND RL ..... 14
**2.5 GAPS AND CONTRIBUTIONS ACROSS STUDIES ..... 15**
2.5.1 FSM'S LACK OF ADAPTABILITY ..... 15
2.5.2 RL'S INEFFICIENCY AND SAFETY CONCERNS ..... 15
2.5.3 CHALLENGES IN TRANSFERRING FROM SIM TO REAL LIFE ..... 15
2.5.4 CONTRIBUTION AND OVERLAPS ..... 16
**2.6 APPLICATION TO THE KANE AND ABEL PROJECT ..... 16**
2.6.1 FSM ..... 16
2.6.2 RL ..... 16
2.6.3 HYBRID IMPLEMENTATION ..... 17
**2.7 CONCLUSION ..... 17**

**CHAPTER 3: PROJECT DESIGN ..... 17**

**3.1 DOMAIN AND TARGET USERS ..... 17**
**3.2 DESIGN CHOICES ..... 18**
3.2.1 FSM AND RL ..... 18
3.2.2 CHOSEN VIDEO GAMES ..... 19
**3.3 STRUCTURE ..... 19**

3.3.1 Super Mario Bros ..... 19
3.3.2 Doom ..... 21
**3.4 Technologies ..... 22**
3.4.1 Python ..... 22
3.4.2 The Farama Foundation Gymnasium ..... 22
3.4.3 gym-super-mario-bros 7.4.0 ..... 22
3.4.4 nes-py ..... 22
3.4.5 ViZDoom ..... 23
3.4.6 PyTorch ..... 23
3.4.7 OpenCV ..... 23
3.4.8 Miscellaneous Technologies ..... 23
**3.5 Timeline ..... 24**
3.5.1 Preliminary Submission ..... 24
3.5.2 Final Submission ..... 24
**3.6 Evaluation and Analysis ..... 25**
3.6.1 Evaluation Metrics ..... 25
3.6.2 Analysis Techniques ..... 25

**CHAPTER 4: IMPLEMENTATION ..... 26**

**4.1 Super Mario Bros. ..... 26**
4.1.1 Finite State Machine ..... 26
4.1.2 Reinforcement Learning Model ..... 28
**4.2 DOOM ..... 30**
4.2.1 Finite State Machine ..... 30
4.2.2 Reinforcement Learning Model ..... 33
**4.3 Hybrid Model ..... 34**
4.3.1 High-Level Vision ..... 34
4.3.2 Finite State Machine ..... 34
4.3.3 Reinforcement Learning ..... 34

**CHAPTER 5: EVALUATION ..... 35**

**5.1 Super Mario Bros. Evaluation ..... 35**
5.1.1 Completion Rate ..... 35
5.1.2 Progression Rate ..... 36
5.1.3 Rewards Accumulation ..... 37
5.1.4 Behavioral Analysis ..... 37
5.1.5 Implementation Effort ..... 37
**5.2 DOOM Evaluation ..... 38**
5.2.1 Completion Rate ..... 38
5.2.2 Progression Rate ..... 39
5.2.3 Rewards Accumulation ..... 40
5.2.4 Behavioral Analysis ..... 40
5.2.5 Implementation Effort ..... 41
**5.3 Hybrid Agent Evaluation ..... 41**

5.3.1 ADAPTABILITY ..... 41
5.3.2 IMPLEMENTATION EFFORT ..... 41

**CHAPTER 6: CONCLUSION ..... 42**

**6.1 SUMMARY OF METHODOLOGIES ..... 42**
**6.2 EVALUATION AND KEY FINDINGS ..... 43**
**6.3 STRENGTHS AND LIMITATIONS ..... 43**
**6.4 IMPLICATIONS ..... 44**
**6.5 RECOMMENDATION FOR FUTURE WORK ..... 44**

**CHAPTER 7: REFERENCES ..... 45**

# CHAPTER 1: INTRODUCTION

## 1.1 Abstract

This project explores the performance, adaptability, and feasibility of two different Artificial Intelligence (AI) methodologies, namely Finite State Machines (FSM) and Reinforcement Learning (RL), for game-playing tasks. The study makes use of video games as controlled yet dynamic testbeds to compare a pre-programmed FSM-based AI with an adaptive RL-based AI. The FSM system relies on a set of predefined rules to act within structured environments, whereas the RL system learns from interaction with the game environment.

The research will try to highlight the strengths and limitations of both approaches in addressing fundamental challenges in AI design, including adaptability and decision-making under uncertainty. Experimental results on the trade-offs between these techniques will provide practical recommendations for game developers, AI researchers, and end-users.

Deliverables are two implementations of AI, one experimental framework, and an extensive performance analysis of the two. The work adds to the growing literature on game AI, providing a comparative study that bridges structured and adaptive methodologies, while having implications for both academic research and practical applications.

## 1.2 Project Template and Idea

This project builds upon the template titled “Kane and Abel: AIs that Play Games”. The original concept was to implement one AI system using pre-programmed behaviors and another using statistical machine learning (ML) techniques. This project extends the template by incorporating a hybrid analysis wherein both methods are applied to different aspects of game-playing, such as decision-making and adaptability. This work differs from the base template on the inclusion of modern evaluation metrics and focuses and end-user experience.

## 1.3 Motivation

Some key motivations for this project are: Game-playing has been a fertile testbed for AI research, serving as a benchmark for the advancement of AI technologies. The structured environment of video games provides a controlled setting to experiment with and evaluate various AI methodologies. Despite the extensive research into the realms of FSMs and RL individually, there is still limited research directly comparing their effectiveness in dynamic, real-time games. By addressing this knowledge gap, the project aims to provide useful insight for game developers, game researchers, and players interested-especially in the tradeoffs between structured and adaptive approaches to AI.

## 1.4 Research Question and Aim

The central research question driving this project is:
"How do Finite State Machines and Reinforcement Learning algorithms compare in terms of adaptability, performance, and user experience when applied to game-playing AI?".

From this question, the following aims are derived:

1. To evaluate strengths and limitations of the FSM-based AI systems in structured game environments.
2. Assess the adaptability of RL-based AI systems in dynamic and unpredictable game scenarios and their efficiency in learning.
3. To identify potential synergies between FSMs and RL, including how hybrid approaches might improve game-playing AI

The proposed objectives are fully aligned with the aims of the project. Two different AI systems will be developed and evaluated to ensure a proper comparison between FSM and RL in the game-playing context. Adaptability underlines one of the important challenges in the design of AI: predictability of behavior should be well-balanced with the capability to behave properly in dynamic and changing situations.

Using video games as the experimental platform is both practical and insightful. Games provide a controlled yet diverse range of challenges, from static puzzles to real-time strategy that can effectively test FSMs and RL. Moreover, the outcomes of this project will have a broader reach than just game playing, as the principles of adaptability and decision-making are at the core of many important applications of AI in robotics and autonomous systems.

## 1.5 Deliverables

The project aims to produce the following:

1. Literature review and analysis
    1. Deep understanding of FSM and RL methodologies, with a key emphasis on their application in game-playing AIs.
    2. Comparative review of the current research output in terms of effectiveness

2. Implementation of Two AI Systems
    1. FSM-based AI system for structured and rule-based decision-making.
    2. RL-based AI capable of learning and adapting game dynamics through trial and error

3. Implementation of the Hybrid model
    1. A proof-of-concept implementation to show that it can function.

4. Experimental Framework and Results
    1. A testbed for benchmarking the two AI systems using video games as the experimental environment.
    2. Quantitative and qualitative analysis of the performance metrics: win rates, adaptability, and computational efficiency.

5. Final Report
    1. Comprehensive documentation of the processes, finding, and conclusions of the project, including recommendations for future research

## 1.6 Domain and Users

This project falls into the domain of game-playing AI - a playground for techniques from AI, given the structured yet challenging environment of many games. Video games are the source of rich, varied scenarios relevant to the real-world decision-making and adaptation. This project targets dynamic game environments where decision-making is required on the fly with strategic adaptability; therefore, this project has close relevance to the greater research area of AI, game development, and human-computer interaction.

The target groups of users of this project are game developers, AI researchers, and the end-users: gamers. To the game developers, this will guide the decision-making on

whether to apply rule-based systems, like FSMs, or adaptive systems, like RL, considering the nature of their games. For example, authors of puzzle or platformer games may find FSMs more convenient, while authors of games with rich, dynamic environments that require more adaptability – such as first-person shooters – will consider RL.

To AI researchers, this project fills the gaps in comparative studies of FSMs and RL. While both methodologies are separately studied, few research works have directly compared their effectiveness in similar scenarios. This project contributes to the understanding of trade-offs, such as ease of implementation, computational costs, and adaptability, offering a foundation for future hybrid approaches.

The gamers indirectly benefit by having more challenging or engaging AI opponents because the insights of this project may inform that design. In addition, developers will understand strengths and weaknesses in FSMs and RL and use those basic knowledge chunks to implement behaviors for either realistic opponents or to improve the overall balance in the game.

(996 words)

# CHAPTER 2: LITERATURE REVIEW

Artificial intelligence is ever in development, finding its usage in robotics, autonomous systems, and game AI. Some of the key techniques being used include Finite State Machines and Reinforcement Learning; both bear their different advantages and challenges. This literature review will look at four main studies that apply FSM, RL, and a fusion of these two. This review therefore discusses the practical and theoretical implications of FSM versus RL and their hybridization by comparing approaches, strengths, and limitations.

## 2.1 Introduction

The field of AI has gone through rapid growth, and decision-making processes have taken center stage, especially in robotics and autonomous systems. FSM and RL are two of the most prominent approaches that enable decision-making. FSM is a well-structured, deterministic framework for tasks that require strict adherence to predefined rules and sequences. On the other hand, RL provides adaptability and optimization heuristics,

making it very strong in dynamic and uncertain environments by learning through trial and error.

While FSM and RL each have their strengths, the limitations of both have motivated the interest in hybrid models that exploit the advantages of both methods. Hybrid approaches merge the robustness and interpretability of FSM with the learning and adaptation capabilities of RL, thereby rendering the systems capable of dealing with complex dynamic scenarios. This literature review discusses such approaches through the analysis of four significant studies applying FSM, RL, or their integration in diverse domains like locomotion, autonomous driving, humanoid robot ingress, and software testing.

This review aims to assess the advantages and limitations of FSM and RL separately and in hybrid configurations concerning their practical applications and theoretical contributions. This review synthesizes insights from these studies to provide a comprehensive understanding of how these techniques can be effectively utilized and improved. The structure of the review follows an overview of FSM and RL, an analysis of selected studies, a comparative discussion of methodologies, the identification of gaps and contributions, and finally, their relevance to AI applications, including game-playing AI.

## 2.2 Background on Finite State Machines and Reinforcement Learning

### 2.2.1 Finite State Machines

A finite state machine represents a deterministic way of modeling, whereby a task is organized into discrete states, joined by predefined transitions. Each state of the FSM denotes an action or a condition, while transitions occur based on well-defined rules or conditions. Its simplicity, interpretability, and reliability are among the most considerable strengths of the FSM. These features make FSM most effective for structured environments and tasks that require rule observance and logical step-by-step processing, like software testing, robotics motion control, and the management of game states.

However, FSM has some remarkable shortcomings. It cannot adapt to either dynamic or unpredictable situations since it relies on pre-defined rules that need to foresee every possible condition. Making FSM scale to complex, real-world settings causes a combinatorial explosion of states, which in turn increases

complexity and reduces scalability. Moreover, FSM does not have any features for learning from experience and/or optimization; thus, its efficiency is severely restricted when applied in environments that involve exploration or adaptation.

### 2.2.2 Reinforcement Learning

Reinforcement Learning is a machine learning paradigm whereby an agent learns a policy by taking actions via interactions in the environment through either a system of rewards or penalties. In RL, agents can perform really well under dynamic and uncertain situations due to the optimality via trial-and-error learning. Popular RL methods such as Proximal Policy Optimization and Deep Q-Networks are now successful across quite varied robotics applications that run to include computer gaming.

The main strength of RL is its flexibility. Unlike FSM, not all states or transitions must be explicitly programmed in RL. Instead, the agent explores the environment to independently learn what the optimal policy is that will maximize the cumulative reward. That makes RL appropriate for tasks where it is hard or impossible to explicitly model a complex and/or unpredictable environment.

RL has several significant challenges despite the advantages. It is a very expensive process computationally, involving huge amounts of data and requiring large computational resources to converge to effective policies. In addition, the learned policies are usually represented by black-box neural networks; hence, interpretability is not achieved by RL. Another concern is safety, because RL agents can behave in unpredictable ways during exploration, which may be unacceptable in high-stakes applications like autonomous driving or healthcare.

### 2.2.3 Significance in Robotics and AI

Both FSM and RL are important for the advancement of robotics and AI. FSM, with its structured approach, is very reliable in safety-critical applications, while RL's adaptability addresses the demands of dynamic and unstructured environments. Hybridization presents a promising solution, combining the deterministic safety and structure of FSM with the exploratory learning capabilities of RL. This synergy enables applications in areas like humanoid robot ingress, autonomous vehicle control, and adaptive testing optimization, where both structure and adaptability are important.

## 2.3 Review of Literature

### 2.3.1 FSM and RL in Quadrupedal Locomotion

The paper by [1] Liu et al. merges FSM and RL to overcome the challenges of dynamic locomotion. FSM manages discrete locomotion states, like stance and swing, for stability and reliability. Reinforcement Learning, mainly PPO, is used within the FSM states to perform an adaptive optimization of low-level controls, like joint angles and forces. Results show robust performance on different terrains, illustrating the structural reliability of FSM and the adaptability of RL.

#### 2.3.1.1 Strengths

The hybrid framework leverages the guaranteed stability from the deterministic state transitions in FSM while enhancing flexibility via RL to cope with unexpected perturbations, such as uneven terrain. Simulated and real-world tests underpin the practical applicability of the system.

#### 2.3.1.2 Limitations

This is limited by the reliance on predefined FSM rules to adapt to entirely novel scenarios. In addition, even the computational demands of RL training within the FSM constraints pose scalability challenges.

#### 2.3.1.3 Applications

This model is useful for robotics that require stable locomotion across many different types of terrains. An example of this is a disaster response robot. Looking deeper, this could also apply in game-playing AIs as games contain multiple environments of varying shapes and sizes.

### 2.3.2 FSM and RL in Autonomous Driving

[2] Hwang et al. incorporated both FSM and RL in their work to address lane-merging challenges while autonomously driving. FSM decomposes the task into phases such as gap selection and the execution of merging, while Policy-Based

Reinforcement Learning optimizes continuous control decisions like steering and acceleration.

#### 2.3.2.1 Strengths

FSM's phase-based structure decomposes complex traffic scenarios into manageable segments that guarantee safety and compliance with rules. RL provides dynamic adaptability for various traffic conditions.

#### 2.3.2.2 Limitations

The FSM structure considers predictable behaviors of traffic, which may not hold in highly unpredictable environments. RL's role is also confined to the optimization of acts within boundaries predefined by FSM, therefore restricting broader learning opportunities.

#### 2.3.2.3 Applications

The approach offers potential for broader application in real-world autonomous driving systems. Its improvement in lane-merging success rates and safety suggests that this method may be suited for predictable but chaotic environments, a setting commonly seen in video games.

### 2.3.3 FSM and RL in Humanoid Robot

In the paper by [3] Wang et al., they used FSM to handle high-level sequencing of ingress steps, such as stepping into the vehicle. On the other hand, Distributed Distributional Deterministic Policy Gradients (D4PG) refines the fine-grained movements like joint adjustments.

#### 2.3.3.1 Strengths

FSM guarantees logical and safe progression of the task, while RL copes with random initial poses and variable vehicle elasticity. It achieves the highest success rate in simulations and proves robust to environmental uncertainties.

#### 2.3.3.2 Limitations

FSF requires heavy manual design and tuning; the confined role of RL limits the full potential for task optimization. Sim-to-real transfer remains one of the main challenges for practical implementation.

#### 2.3.3.3 Applications

This approach allows humanoid robots to perform complex tasks in semi-structured environments. Some examples include industrial automation and service robotics. Games on the other hand come with varying environments. Some are structured and others are chaotic. The approach discussed in this paper could be applied in video games but in specific situations like movement or navigation.

### 2.3.4 FSM and RL in Testing Optimization

The paper by [4] Türker et al. uses RL to prioritize FSM state transitions in software testing. In return, this reduces redundant paths and improves efficiency.

#### 2.3.4.1 Strengths

RL intelligently explores high-priority transitions, optimizing FSM-based testing workflows. This reduces computational costs while maintaining a comprehensive test coverage.

#### 2.3.4.2 Limitations

This approach heavily relies on structures defined by the FSM. Therefore, applying this approach to less structured testing scenarios may not be appropriate. Exploration by RL is confined to the states predefined by the FSM, reducing its adaptability.

#### 2.3.4.3 Applications

This method is effective in software testing domains that require efficiency with no compromise on coverage, such as automated regression testing.

## 2.4 Comparative Analysis of FSM and RL

### 2.4.1 Strengths and Weaknesses of FSM and RL

FSM and RL each bring their strong and weak points to applications in AI. FSM finds its best application in highly structured tasks where safety can be critical, given their nature of being deterministic because rules are followed to execute tasks logically. It's highly interpretable, especially in areas where explainability is key, such as self-driving cars or test optimizations. However, FSM cannot adapt to dynamic environments, and complex tasks result in a combinatorial explosion of states.

On the other hand, RL does extremely well in dynamic and uncertain settings by learning the best strategy through iterative interactions with the environment. The adaptation to tasks is accomplished when explicit modeling is infeasible. However, RL generally faces the problems of computational inefficiency, lack of interpretability, and safety risks during exploration. These issues render it unsuitable for use in applications related to safety-critical settings without any further constraints.

### 2.4.2 Hybrid Approaches

The hybridization of FSM and RL addresses the shortcomings of each method. On one hand, FSM provides structure and reliability: it defines task phases and ensures logical transitions between them. On the other hand, RL optimizes decision-making within those phases. This synergy is apparent in applications like autonomous driving and humanoid robot ingress, where FSM guarantees safety and progression of the task, while RL adapts to real-time variability.

While hybrid approaches show better performance and safety, they introduce complexity into design and implementation. Careful tuning of FSM constraints with RL's optimization capabilities requires domain expertise. Scalability remains a challenge, too, especially in highly dynamic or unpredictable environments.

### 2.4.3 Individual use of FSM and RL

FSM alone works for static, rule-driven tasks where adaptability is not of importance, such as predefined testing workflows or highly controlled

environments. On the other hand, RL alone is good in exploratory tasks or dynamic scenarios, like game AI, where adaptability and learning are crucial. However, the unstructured nature of RL can easily lead to suboptimal solutions in tasks that require strict adherence to rules.

## 2.5 Gaps and Contributions Across Studies

These reviewed works provide valuable insights into FSM, RL, and their hybridization; however, there are several gaps that remain unaddressed:

### 2.5.1 FSM's Lack of Adaptability

FSM is challenged in providing performance in dynamic or unstructured environments since it relies on a priori-defined states and transitions. In the face of unexpected situations, though FSM guarantees safety and rule observance, it is not flexible. For instance, the FSM in the humanoid ingress task required very much hand-tuning and was therefore hardly scalable. Obtaining this will involve designing adaptive mechanisms within FSM frameworks or hybrid approaches.

### 2.5.2 RL's Inefficiency and Safety Concerns

While effective to learn an optimal policy, the trial-and-error nature of RL is inherently computationally expensive and provides no safety guarantees. Consider autonomous driving: currently, RL tries to mitigate such risks simply by being confined to optimization over an action given a provided FSM constraint. These challenges can thus be ameliorated with improved efficiencies in reinforcement learning and possible safety measures using either reward shaping or constrained exploration.

### 2.5.3 Challenges in Transferring from Sim to Real Life

Most research works great in simulation but usually struggles to find out how to transfer the results to the real world. Both the humanoid ingress and quadrupedal locomotion studies showed that the gap existed in hybrid model performance from simulation to real-world deployment. Basic research in areas like transfer

learning techniques or domain randomization will be of much help in bridging the gap.

### 2.5.4 Contribution and Overlaps

These reviewed works complement each other with respect to specific challenges: for instance, the major drawback of RL-unpredictability-is outbalanced by the structural reliability of FSM, whereas RL serves to enhance the rigidity of FSM. Such a promising combined application of robustness and adaptability can be shown in hybrid models of the two protocols, such as quadrupedal locomotion and autonomous driving. However, this is something to be generalized to wider applications in the future.

## 2.6 Application to the Kane and Abel Project

### 2.6.1 FSM

FSM models the scope of game states and transitions-like attack, defend, and even collecting resources. In general, it enforces the logic of a game with sheer clarity of rules and hence predictable AI behaviors. For example, FSM will manage multi-phased strategies, initiating transitions between states when particular conditions occur to maintain consistent game flow.

### 2.6.2 RL

The various exploratory strategies require that the RL be sufficient on its own. It learns to counter opponents, maximize rewards, and make adaptations in real-time in dynamic environments. For instance, in AI, it allows refining micro-strategies in adjusting resource allocations or optimizing moves as would best fit an opponent's move.

### 2.6.3 Hybrid Implementation

They all work best when put together: FSM lays the basic foundations of decision-making and rules, while RL optimizes the actions for the states. In all, it allows AI to act accordingly in game mechanics while improving its adaptability. For instance, FSM directs the decisions of transitioning to attack or to defense, while RL refines the decisions according to opponents or resources.

## 2.7 Conclusion

FSM brings in the structure, reliability, and interpretability, including safety-critical applications, while RL dynamically adapts and optimizes. Both show major drawbacks: inflexibility in the case of FSM and safety issues because of inefficiency in the case of RL. Hybrid methods combine the structure provided by FSMs with the adaptability enabled by RL. Several applications reported good performance for locomotion, autonomous driving, and software testing. However, challenges concerning scalability, sim-to-real transfer, and RL safety remain open.

In most AI game projects, FSM handles all the rules and transitions, while RL empowers them toward better adaptability. Eventually, they come together to make strong AI systems that eventually upgrade the game. Future work will consider scalability, RL safety, and modular hybrid design for efficacy, robustness, and domain adaptability.

(2428 words)

# CHAPTER 3: PROJECT DESIGN

## 3.1 Domain and Target Users

The project brings together artificial intelligence and game-playing, working in the development and evaluation of intelligent game-playing systems. The work specifically compares FSMs with RL to establish intelligent systems that can be in relation with game environments. It involves AI techniques that use rule-driven logic and dynamic adaptability, which form part of game design for modern games.

Its main users will be game developers, AI researchers, and gamers. Game developers will learn how to use FSM and RL to create intelligent game characters and improve engagement with players. The project demonstrates to AI researchers how FSM and RL can complement each other, overcoming the rigidity of FSM and the inefficiency of RL. Teachers can use it to convey AI concepts, pointing out adaptation because different games require different techniques.

The project focuses on finding a balance between structured decision-making and adaptability of AI systems in interactive games. FSM brings deterministic logic for rule adherence and predictable behavior, while RL focuses on dynamic adaptability and learning optimal strategies through trial and error. This project improves AI approaches and enriches game development by addressing user expectations and game demands. Focusing on FSM and RL, and their hybridization, it forms a foundation for novel AI design in digital media.

## 3.2 Design Choices

### 3.2.1 FSM and RL

FSMs and RL are two of the most widely used methods for designing intelligent systems in game AI; each has its own advantages. FSMs provide a clear decision-making framework with structured state transitions, ensuring totally predictable behavior according to the rules—something that is very important in terms of logical consistency in games. FSMs model even the smallest details in game elements, like enemy behavior or player interactions, as clear transitions are good for a better user experience and for keeping the balance.

RL is a dynamic approach that enables AI to train optimal strategies by interacting with the environment. Compared with FSM, RL performs remarkably well in exploration and adaptive problems, especially under the competitive games. It is used to adapt decision-making based on feedback, enabling AI to handle complex and unpredictable situations—ideal for variable games. However, its stochastic nature and high computational needs limit its use in safety-critical or rule-intensive scenarios. For this project, we will make use the Proximal Policy Optimization (PPO) algorithm, a method known for its quicker training times and better outputs.

The hybridization of FSM and RL will combine the strengths of both to make it a robust and adaptive system. On the high level, FSM keeps track of states like "attack" or "defend" through predefined rules, while RL optimizes the actions within those states. This combination overcomes the rigidity of FSM and the

inefficiency of RL by leveraging the structured decision of FSM with adaptive learning of RL. This project meets the need of game developers and researchers for applied, scalable AI solutions using FSM and RL. In this way, it assures that intelligent systems can achieve a balance between rule conformance and adaptability in the most varied gaming environments.

### 3.2.2 Chosen Video Games

#### 1. Super Mario Bros

Super Mario Bros (SMB) is a structured platformer-typer video game with clear rules and distinct objectives. This would be a perfect fit for FSM as it thrives on explicit state transitions. RL, on the other hand could optimize in-game actions like jump timing, enemy interactions, and come up with a strategy like humans do.

#### 2. Doom

Doom provides a high-speed first-person environment that focuses on uncertainty and fast adaptation. Unlike SMB, it focuses on survival, resource management, and efficiency in combat, making it much more suitable for testing the adaptability of RL. While FSM can model basic actions like moving or attacking, its limitations would become very clear in the dynamic scenarios of Doom.

## 3.3 Structure

### 3.3.1 Super Mario Bros

#### 3.3.1.1 FSM Architecture

FSM is used to define high-level game states and transitions. To simplify the development and evaluation, we will not be making use of the full action space as it contains 12 actions. Instead, we will focus on 5. For SMB, the FSM structure includes the following states:

- **IDLE**: (No Operation) Agent would stand still and no action is required
- **MOVE_RIGHT**: Agent moves forward (to the right), usually when the path is clear

- **JUMP_RIGHT**: The agent jumps while moving to the right, usually to overcome an obstacle, jump over or onto an enemy, or jump to get to a higher platform

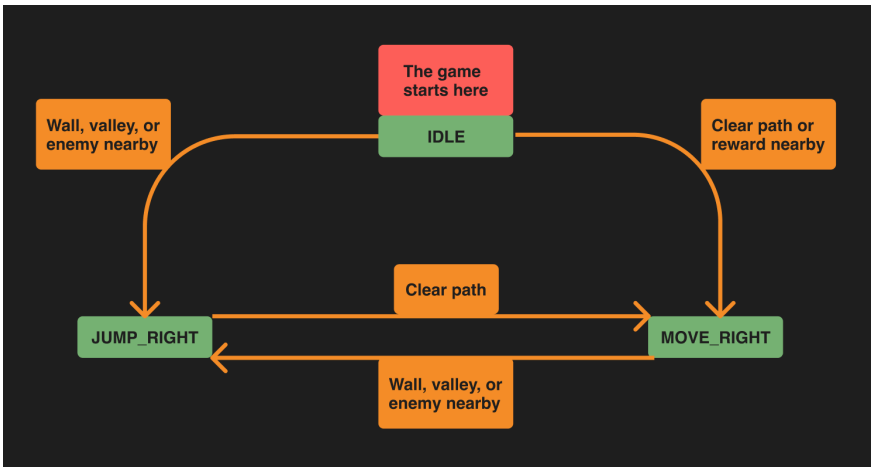



**Figure 3.1. State transition diagram for Super Mario Bros.**

### 3.3.1.2 RL Architecture

For the RL model, we will make use of the PPO algorithm. For SMB, the agent will learn policies to maximize rewards by interacting with the environment. We will be using the following reward structure:

- +10 for progressing forwards to encourage progression
- +50 for defeating enemies to encourage defeating enemies, a tough task
- +100 for completing levels to encourage level completion
- -5 for idling for more than a second to discourage idling
- -50 to discourage death

It is important to note that this structure will serve as a starting point and would change as development progresses. The rewards structure will also change as priorities change over time like prioritizing high scores or faster level completion.

## 3.3.2 Doom

### 3.3.2.1 FSM Architecture

For Doom, we will focus on two states:

- **SEARCH**: The agent scans the environment, usually to look for enemies or resources
- **ATTACK**: The agent engages in combat when an enemy is detected, usually when the agent has sufficient health
- **AIM**: The agent adjusts its crosshair onto the nearest enemy

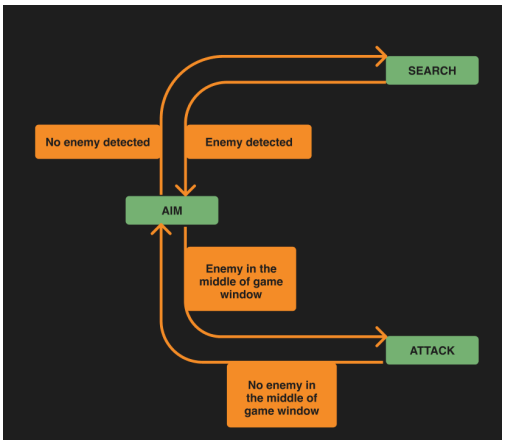



**Figure 3.2. State transition diagram for DOOM**

### 3.3.2.2 RL Architecture

For the RL model, we will still make use of the PPO algorithm. This is to ensure a fair and more accurate comparison and evaluation. We will be using the following rewards structure:

- +50 for defeating enemies to encourage combat encounters
- -10 for taking damage to discourage taking damage
- -50 for dying to discourage death

## 3.4 Technologies

The tools, frameworks, and programming languages used in this project are one of the common and modern ways to implement and evaluate AI game-playing agents in gaming environments. The technologies selected have their bases founded on being robust, compatible, and widely adopted in AI and game development. Details of the technologies used are written in this section.

### 3.4.1 Python

Python will be the core language for this project as it is a versatile language, strong in support for machine learning and game development. Its clean syntax and rich ecosystem empower quick AI agent development. It is also compatible with a multitude of machine learning frameworks, making development and integration seamless. On top of that, its high-level nature allows for quicker and simplified development as the language handles the inner workings of development, allowing us to focus on high-level development.

### 3.4.2 The Farama Foundation Gymnasium

Gymnasium provides a standard interface from which the training and testing of reinforcement learning agents are usually done. It is a maintained fork of the original, but now depreciated, OpenAI Gym. It includes a collection of reinforcement learning environments that are designed for reinforcement learning research. This ensures compatibility with algorithms implemented with PyTorch. The Gymnasium library would ease the task of interacting and benchmarking AI agents against standardized environments.

### 3.4.3 gym-super-mario-bros 7.4.0

This package is based on OpenAI Gym but is cross compatible with Gymnasium. It provides a structured environment for game-playing AI implementation and evaluation. The package makes use of the original Super Mario Bros game file from the Nintendo Entertainment System (NES). Due to this, its environments, physics, and logic is as accurate as the real game itself.

### 3.4.4 nes-py

The nes-py library provides a lightweight NES game emulator onto which the gym-super-mario-bros environment is based. Its effective integration with Python allows fine-grained control on in-game events. Applying nes-py, this project will

be able to provide fine granularity into the manipulation of the environment and, as such, will facilitate and simplify the design of both FSM and RL models.

### 3.4.5 ViZDoom

ViZDoom is a 3D gaming environment used to improve and assess AI game-playing agents' performance in a first-person shooter (FPS) environments. ViZDoom adds a fully dynamic and complex game logic, like navigation and combat with real-time decisions. Its flexibility with custom-designed scenarios and with Python make it a perfect testbed for RL integration.

### 3.4.6 PyTorch

PyTorch is one of the leading deep learning frameworks used throughout implementations of reinforcement learning algorithms. With its dynamic computation graph, ease of debugging, and rich ecosystem of prebuilt tools and models, PyTorch is an excellent chouce for training and deploying RL agents. Its integration with Gymnasium environments gives rise to easy implementations of RL methods.

### 3.4.7 OpenCV

This project applies OpenCV for image processing and computer vision. It is helpful in pre-processing raw frames from Gymnasium environments. It can do tasks like resizing, grayscaling, thresholding, and frame stacking efficiently. OpenCV will be the eyes of our FSM and RL models, allowing these agents to "see" the game and act accordingly.

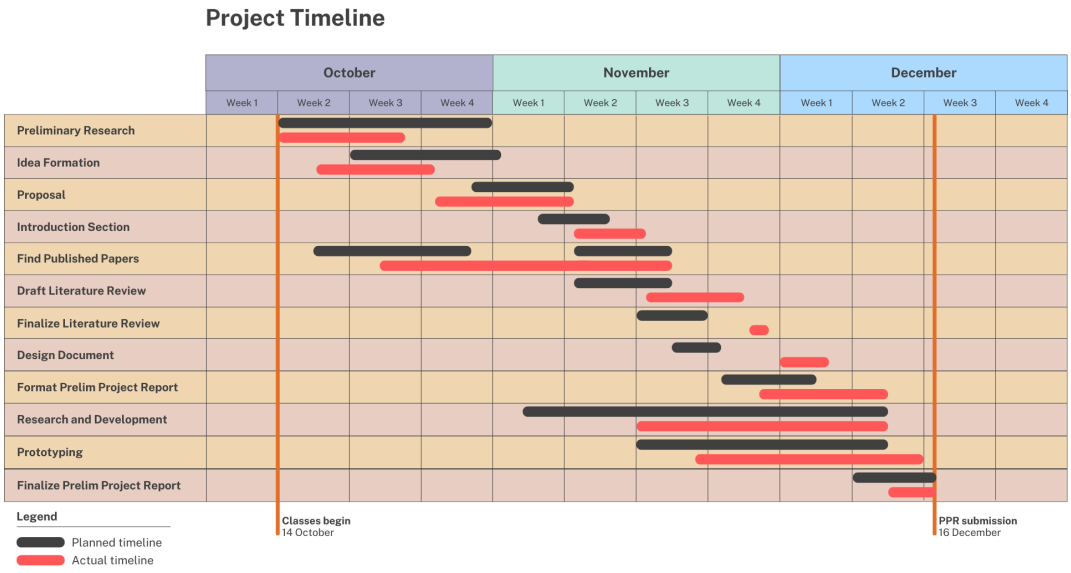
### 3.4.8 Miscellaneous Technologies

While the above-mentioned technologies represent the backbone of this project, other tools may be used during further development and evaluation. Among other libraries, Matplotlib will be used with regards to data visualization for performance metrics analysis and presentations. On top of that, Git and GitHub will be used for version control and repository management respectively.

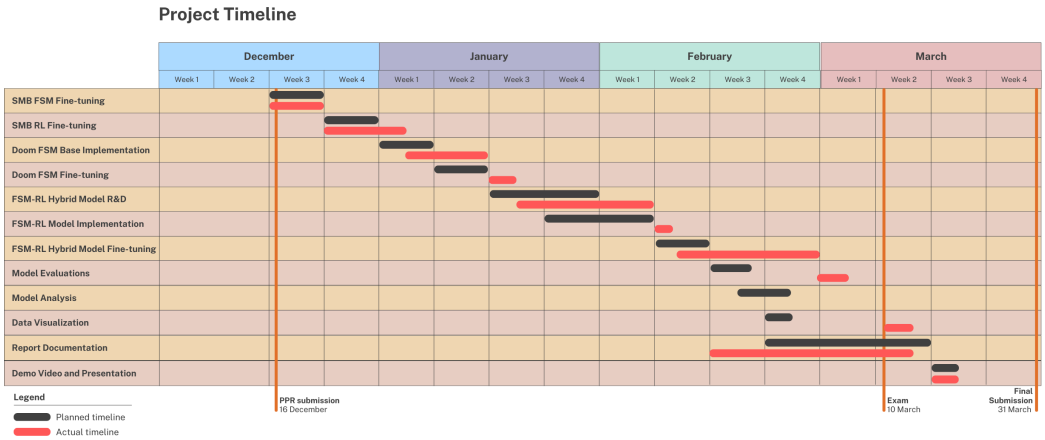
## 3.5 Timeline

The figures below outline the timeline for the project. It has been split into two parts, preliminary and final submission, for ease of viewing.

### 3.5.1 Preliminary Submission




### 3.5.2 Final Submission



## 3.6 Evaluation and Analysis

### 3.6.1 Evaluation Metrics

Both FSM and RL models will be evaluated based on metrics concerning how well the models can play the game. The metrics are as follows:

1. Completion Rate

   The completion rate refers to the ratio of the levels completed by the model to all the levels attempted.

2. Progression Rate

   The percentage of a level progressed by the model.

3. Rewards Accumulation

   The rewards accumulated during game play will give us a view regarding the models' efficiency.

### 3.6.2 Analysis Techniques

1. Comparative Performance Analysis

   Both models are tested on the same set of levels, and their metrics are compared against each other to highlight strengths and weaknesses.

2. Temporal Analysis

   Assessing how the models perform as time progresses will shed light on their relative stabilities and adaptability.

3. Behavioral Analysis

   This involves observing and interpreting the models for certain gameplay behaviors.

(1741 words)

# CHAPTER 4: IMPLEMENTATION

## 4.1 Super Mario Bros.

### 4.1.1 Finite State Machine

The implementation of this model involves three states: IDLE, MOVE-RIGHT, and JUMP-RIGHT. These states are modelled from the three main controls used to navigate through most of the earlier levels. Their definitions are as follows:

- **IDLE**: This is the beginning state. No keys are activated here. Once the agent transitions out of this state, it will never be revisited again.
- **MOVE-RIGHT**: This acts as the default state as the main action required to progress through levels is to move right. This state gets transitioned into when the agent detects no nearby threats.
- **JUMP-RIGHT**: In this state, the agent will be jumping to the right. Transitioning into this state requires the agent to detect a nearby threat, like enemies, or a nearby obstacle, like pipes and stairs.

#### 4.1.1.1 Vision

The agent's vision was processed using thresholding and contours. Since the game is modelled in 2D, edge detection isn't necessary as using contours would already detect the edges after thresholding. On top of these visual processing methods, we used a variant of the environment that removes the necessary background image from the levels to simplify visual processing. Figure 4.1 shows the unprocessed image seen my humans whereas figure 4.2 shows the agent's vision after image processing.

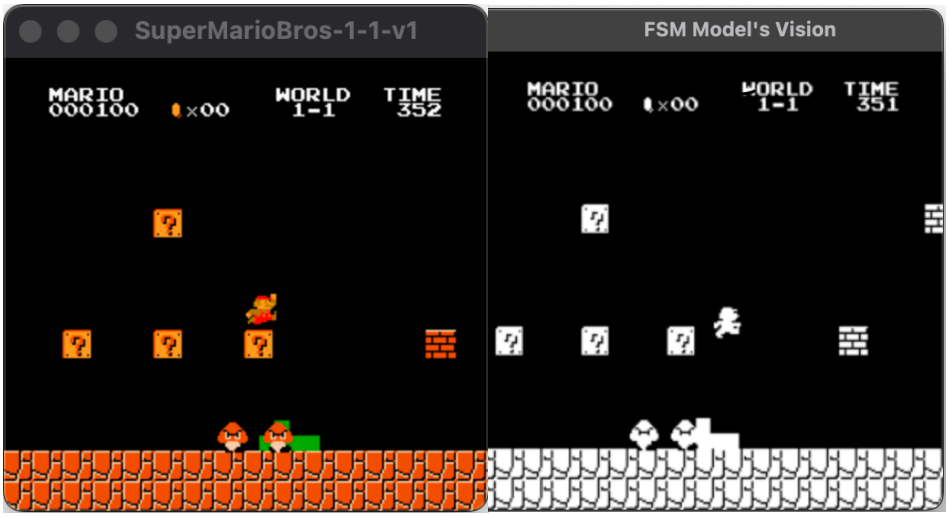



**Figure 4.1. Unprocessed state**




**Figure 4.2 . Processed state**

#### 4.1.1.2 States

Ignoring the initializing state (IDLE), there are only two states: MOVE-RIGHT and JUMP-RIGHT. The decision to model states from the action space was made due to the complexity of the environment. With many events occurring simultaneously in one frame, confusion starts to arise in a more complicated model. For instance, in one processed frame, the agent could be in danger from an enemy, the agent may be over a valley after jumping previously, and an enemy projectile may be coming toward the agent. With this depth of plausible events, especially on harder game levels, an FSM starts to get complicated quickly and it would not be able to account for all the plausible events and thus, states were heavily simplified. Pipes, valleys, and enemies are classified as obstacles and tied to the JUMP-RIGHT state whereas the MOVE-RIGHT state is activated only when there are no nearby obstacles.

#### 4.1.1.3 State Transitions

A state transition table, shown in figure 4.3, is created to simplify the events mentioned in section 4.1.1.2. It accounts for the aforementioned obstacles and explicitly lists possible state transitions so that the agent does not go into an endless loop. The table is modelled from the diagram shown in figure 3.1.

```
self.transitions = {
    "IDLE": {
        "ENEMY": "JUMP-RIGHT",
        "PIPE": "JUMP-RIGHT",
        "VALLEY": "JUMP-RIGHT",
        "DEFAULT": "MOVE-RIGHT"
    },
    "MOVE-RIGHT": {
        "ENEMY": "JUMP-RIGHT",
        "PIPE": "JUMP-RIGHT",
        "VALLEY": "JUMP-RIGHT",
        "DEFAULT": "MOVE-RIGHT"
    },
    "JUMP-RIGHT": {
        "ENEMY": "JUMP-RIGHT",
        "PIPE": "JUMP-RIGHT",
        "VALLEY": "JUMP-RIGHT",
        "DEFAULT": "MOVE-RIGHT"
    }
}
```

**Figure 4.3. State transition table**

### 4.1.2 Reinforcement Learning Model

The implementation of this model required many iterations of experimentation due to the balancing of the reward system, and mainly, hyperparameter tuning. On top of that, training is a time consuming and computationally expensive process so generally, the implementation of the RL model took exponentially longer than that of the FSM model.

We made use of various environment wrappers to allow the PPO model to receive more detail about its environment. These wrappers are:

- **GrayscaleObservation**: To reduce the number channels as RGB channels are redundant. Also used to reduce input size, allowing for more efficient training.
- **ResizeObservation**: To also reduce input size and allow for more efficient feature extraction.
- **SubprocVecEnv**: To create vectorized environments with the aid of subprocesses for parallelized training.
- **VecFrameStack**: To provide temporal context to the model. It aids the model to infer motion from frame sequences. We used a stack of 4 frames.
- **DummyVecEnv**: This is only used during testing, to create a vectorized environment on a single process.
- **VecMonitor**: Used to monitor and log episode statistics.

We also created custom wrappers:

- **CustomJoypadSpace**: This fixes compatibility issues with the reset() method by removing the "seed" argument.
- **SkipFrame**: Combines multiple consecutive frames. Not all frames provide useful information. Makes training more efficient by focusing on major rather than minor changes between frames.
- **CustomReward**: Modifies the reward signal by implementing our custom rewards.

#### 4.1.2.1 Vision

A custom convolutional neural network (CNN) was designed to tailor to the game. The default CNN used in the PPO uses three layers. Our network makes use of four layers to extract visual data from the screen. Figure 4.4 shows the code that makes up the custom CNN.

```
class VisionFeatureExtractor(BaseFeaturesExtractor):
    def __init__(self, observation_space: gym.spaces.Box, features_dim):
        super(VisionFeatureExtractor, self).__init__(observation_space, features_dim)
        n_input_channels = observation_space.shape[0]
        self.cnn = nn.Sequential(
            nn.Conv2d(n_input_channels, 32, kernel_size=3, stride=2, padding=1),
            nn.ReLU(),
            nn.Conv2d(32, 32, kernel_size=3, stride=2, padding=1),
            nn.ReLU(),
            nn.Conv2d(32, 32, kernel_size=3, stride=2, padding=1),
            nn.ReLU(),
            nn.Conv2d(32, 32, kernel_size=3, stride=2, padding=1),
            nn.ReLU(),
            nn.Flatten(),
        )

        # Compute shape by doing one forward pass
        with th.no_grad():
            n_flatten = self.cnn(th.as_tensor(observation_space.sample()[None]).float()).shape[1]

        self.linear = nn.Sequential(nn.Linear(n_flatten, features_dim), nn.ReLU())

    def forward(self, observations: th.Tensor) -> th.Tensor:
        return self.linear(self.cnn(observations))
```

**Figure 4.4. Custom CNN code**

#### 4.1.2.2 Rewards

A custom reward structure was used to prioritize completing the level over to collecting collectibles. This was done to speed up training time and leave lesser room for complications. Higher rewards and penalties were given to certain properties we would like the agent to prioritize.

| Event | Reward |
|---|---|
| Progressing rightward | +x distance travelled |
| Completing the level | +100 |
| Losing a life | -15 for each life lost |
| Clock ticks | -1 for every second where level is not completed |
| Coins | +5 for each coin collected |
| Death | -500 |
| Level time limit exceeded | -500 |

**Figure 4.4. Reward structure for the PPO model**

## 4.2 DOOM

### 4.2.1 Finite State Machine

The implementation of the model follows three states: SEARCH, AIM, and ATTACK. The following points describes the purposes and actions taken within the specified states:

- **SEARCH**: This is the default state. The agent stays in this state until an enemy is detected. The agent searches for enemies by constantly moving or turning left (depending on the map being played).
- **AIM**: This state takes care of aiming the crosshair of the player onto the enemy. It will prioritize aiming at the enemy nearest to the center of the game window. The agent stays in this state as long as an enemy is detected and that no enemies are on the agent's crosshair.
- **ATTACK**: The agent enters this state when an enemy is detected on the crosshair. A grace of 5px is given to balance accuracy and consistency as the crosshair does not need to be in the center of the enemy to attack it. Allowing for a small grace amount also helps the agent attack enemies that are further away.

#### 4.2.1.1 Vision

The model's vision comprises of four preprocessing layers: grayscaling, median blur, edge detection, and contours. There was a tough decision to be made when deciding between median and gaussian blur, but we decided on median blur as it is more consistent at detecting enemies whereas gaussian blur detects enemies more accurately, but its detection would not persist, and it would flicker constantly. The table below (figure 4.5) shows that comparison between both methods. Figure 4.6 below shows the difference between using different blur methods. The

differences may be minimal, but it affects the performance of the model in the long run.

| Blur Method | Clarity | Stability | Detection Persistency | Accuracy |
|---|---|---|---|---|
| Median | Less clear (thicker edge borders) | More stable | Persistent detection | Less accurate. Due to a simpler blur method, it occasionally detects random edges. |
| Gaussian | Very clear edges | Less stable | Prone to flickering | Very accurate |

**Figure 4.5**

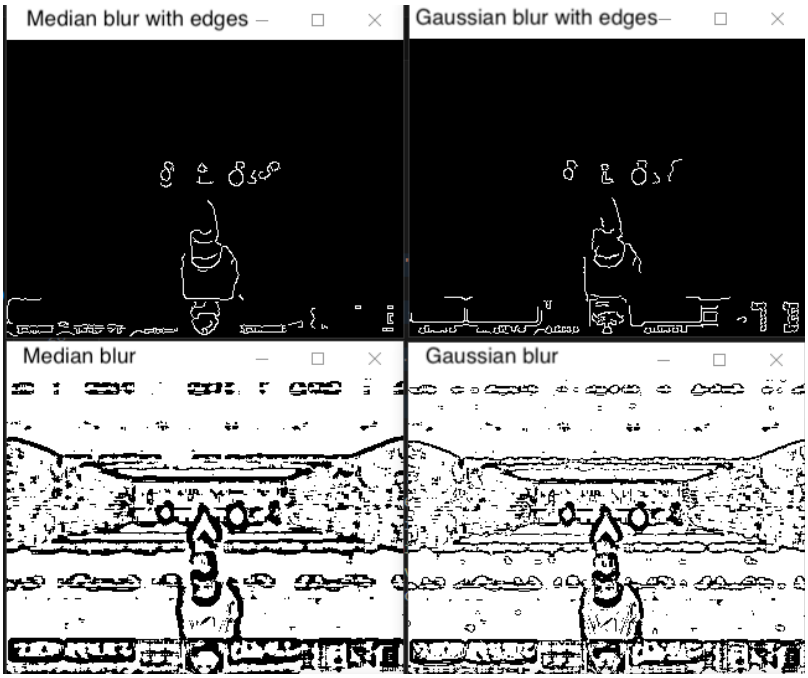



**Figure 4.6**

#### 4.2.1.2 States

ViZDoom offers highly customizable environments allowing end users to control low level game variables such as camera angle, reward structure, game difficulty, and even access to depth and audio buffers. Due to this, we could program states to focus on specific actions like aiming.

#### 4.2.1.3 State Transitions

Unlike the approach to the Super Mario Bros. FSM, we explored integrating both the environment and FSM into a single class rather than creating two classes and later merging them. Due to the simplicity of the environment and states, we used simple if-else statements to control state transitions.

```
def transition(self):
    contours = self.detect_enemies()
    self.enemy_detected = True if len(contours) > 0 else False
    print(f"Current state: {self.state}")

    if self.state == "SEARCH":
        # Enemy detected: switch to AIM state
        if self.enemy_detected:
            self.state = "AIM"
            print("Switching to AIM state")

    elif self.state == "AIM":
        # No enemies detected: switch to SEARCH state
        if not self.enemy_detected:
            self.state = "SEARCH"
            print("Switching to SEARCH state")

        # Enemy in the center of the screen: switch to ATTACK state
        elif self.enemy_centered:
            self.state = "ATTACK"
            print("Switching to ATTACK state")

    elif self.state == "ATTACK":
        # No enemies in the center of the screen: switch to SEARCH state
        if not self.enemy_centered:
            self.state = "AIM"
            print("Switching to AIM state")
```

**Figure 4.7. Transition method in FSM class.**

#### 4.2.1.4 Ensuring Adaptability

Due to each map being focused on different aspects of the game, we chose three maps to model the FSM on as they have similar end goals and action spaces. The maps used are basic, defend_the_center, and defend_the_line. The only difference, in terms of action space, between these maps is between basic and the other two. Basic's action space is MOVE_LEFT, MOVE_RIGHT, and ATTACK whereas for the other two, they have TURN_LEFT and TURN_RIGHT in place of basic's first two actions.

### 4.2.2 Reinforcement Learning Model

Implementing this model potentially could have been made easier by using ViZDoom's depth buffer and automatic object labelling. However, since these features are not usually available to end users of video games, we decided to go with another approach. Similarly to Super Mario Bros. RL agent, developing this required multiple iterations of changes namely due to hyperparameter tuning and reward shaping. Also in this implementation, we focus on simplicity rather than modularity of code. As compared to the Super Mario Bros. RL agent which includes a modular approach to each problem by using environment wrappers, we incorporated everything into a singular class.

#### 4.2.2.1 Vision

We created a custom vision wrapper to be used in place of the default convolutional neural network model packaged with Stable Baselines 3's PPO model. The source code is the same, but we did this to simplify the development for future iterations of this project in the event we need to modify the CNN.

#### 4.2.2.2 Rewards

Reward shaping for ViZDoom is relatively simple as compared to Super Mario Bros. Each map provides a unique set of rewards, catered toward the end goal of the map. On top of this default reward structure, we incorporated our own to further improve on it. We ignored a death penalty as in the training map, death is inevitable due to limited ammo.

| Event | ViZDoom's Rewards | Our Rewards (added on) |
|---|---|---|
| Killing enemy | +1 | +10 |
| Death | -1 | nil |
| Wasting ammo | nil | -3 per bullet |

**Figure 4.8. Reward structure for training**

#### 4.2.2.3 Training

Training was done on the defend_the_center map as this is technically the most difficult amongst the three that were selected. The theory is that the model should be quite adaptable between the three maps since they share similar action spaces. However, the varying textures present in the three environments may impact the performance. The model was ultimately trained on one million timesteps.

## 4.3 Hybrid Model

Due to the nature of this hybrid concept, we took a modular approach to its development. We chose to implement this experimental agent on the deadly_corridor map as its goal is straightforward: to reach the end of the corridor safely. Here, our focus is to simplify the environment as much as possible so that we can test and verify that this concept can work.

### 4.3.1 High-Level Vision

To simplify development, ViZDoom's label buffer was used. This buffer is an array of visible entities on screen. We then filter out the on-screen entities and only allow enemies to pass through. In practical, some form of computer vision technique would be used here but for the purposes of proving the viability of this hybrid concept, we stick to using the label buffer.

### 4.3.2 Finite State Machine

This acts as a high-level decision maker. We introduce two main states: SEARCH and ATTACK. Their descriptions are as follows:

- **SEARCH**: This is the default state where the agent navigates through the hallway.
- **ATTACK**: This state is activated when enemies are detected. The agent will focus on killing enemies in this state until no more are seen.

When a state is activated, it selects the RL model that was trained for that state for further decision making.

### 4.3.3 Reinforcement Learning

Two models are trained for the two FSM states. The SEARCH model was trained by giving it access to movement actions only. The default map's reward structure was used here, and it was trained on 200,000 timesteps. Figure 4.9 demonstrates the reward structure.

The ATTACK model was trained by giving it access to attack actions as well as turning left and right for aiming purposes. Unlike the SEARCH model, the default reward structure used was modified to accommodate to this model's specific goal of killing enemies. Figure 4.10 shows the reward structure.

| Event | Reward |
|---|---|
| Moving closer to the vest/end of corridor | +difference in x position |
| Moving further from the vest/end of corridor | -difference in x position |
| Death | -100 |

**Figure 4.9. Reward structure for SEARCH model**

| Event | Reward |
|---|---|
| Wasting ammo | -0.5 per bullet |
| Killing enemies | +10 per kill |
| Each timestep when 2 enemies have not been killed | -0.01 per timestep |

**Figure 4.10. Reward structure for ATTACK model**

(1890 words)

# CHAPTER 5: EVALUATION

## 5.1 Super Mario Bros. Evaluation

### 5.1.1 Completion Rate

The FSM agent didn't manage to complete any of the levels due to the increased complexity of obstacles as the agent progresses the level. Unsurprisingly, the RL agent managed to complete the first level: the level it was trained on. However, on higher levels, it did not manage to complete any despite testing more than 100 episodes. Figure 5.1 shows a table of completion rates of each agent in each level. Overall, the RL agent prevails.

| level | 1-1 | 1-2 | 1-3 | 1-4 | 2-1 | 2-2 | 2-3 | 2-4 | 3-1 | 3-2 | 3-3 | 3-4 | 4-1 | 4-2 | 4-3 | 4-4 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| RL - avg level completed (%) | 71.54 | 27.1 | 16.68 | 12.16 | 22.7 | 43.48 | 26.15 | 12.8 | 17.58 | 18.53 | 12.89 | 11.4 | 29.33 | 15.5 | 17.08 | 13.18 |
| RL - furthest (%) | 100.0 | 41.75 | 24.2 | 19.1 | 37.52 | 60.85 | 37.63 | 17.28 | 25.85 | 44.18 | 14.39 | 14.94 | 53.91 | 30.12 | 25.06 | 28.59 |
| FSM - level completed (%) | 63.59 | 10.95 | 9.02 | 7.88 | 16.77 | 19.4 | 3.49 | 11.81 | 18.79 | 19.6 | 9.02 | 9.43 | 13.87 | 12.17 | 13.56 | 3.89 |

| level | 5-1 | 5-2 | 5-3 | 5-4 | 6-1 | 6-2 | 6-3 | 6-4 | 7-1 | 7-2 | 7-3 | 7-4 | 8-1 | 8-2 | 8-3 | 8-4 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| RL - avg level completed (%) | 12.81 | 26.11 | 14.61 | 11.0 | 13.5 | 16.65 | 12.53 | 10.55 | 14.8 | 20.38 | 19.29 | 8.1 | 8.34 | 9.72 | 16.23 | 3.22 |
| RL - furthest (%) | 22.84 | 34.2 | 40.05 | 17.11 | 37.52 | 25.81 | 19.93 | 15.49 | 27.85 | 35.11 | 29.32 | 13.34 | 25.13 | 10.21 | 26.95 | 3.85 |
| FSM - level completed (%) | 8.26 | 11.52 | 9.02 | 11.81 | 12.25 | 12.7 | 11.62 | 7.88 | 22.55 | 19.4 | 3.49 | 8.73 | 3.48 | 7.64 | 11.61 | 3.13 |

**Figure 5.1. Table of data showing agent progression through levels**

## 5.1.2 Progression Rate

With reference to figure 5.1, we can see that the FSM agent averages at approximately a 12.76% completion rate per level. The RL agent, on the other hand, manages to peak at a 30% average. This shows the agent's ability to somewhat adapt to new environments.

Apart from the first level, the RL agent can still do well on later some levels. The FSM agent however shows that it lacks the means to adapt to new environments. This is evident when the data is plotted on a smoothened graph (figure 5.4). It shows the FSM agent quickly heading to a plateau and fluctuates slightly around the plateau. In the smoothened graph, it can be deduced that the RL agent gradually gets less adaptable as level difficulty increases.

| | |
|---|---|
| **RL - level completed average (%)** | 18.31 |
| **RL - furthest average (%)** | 30 |
| **FSM - level completed average (%)** | 12.76 |

**Figure 5.2. Table of data showing the average agent progression through levels**

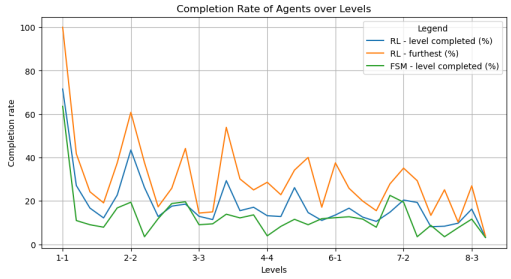



**Figure 5.3. Graph of completion rates**

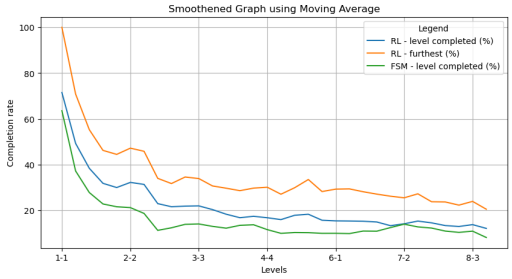



**Figure 5.4. Smoothened graph of completion rates**

### 5.1.3 Rewards Accumulation

The FSM model was unable to collect any rewards before each death but was able to successfully attack three enemies before its death. The RL model on the other hand manages to collect rewards. The figure below (figure 5.5) shows its averaged rewards over every 50,000 timesteps during its training. The graph slowly coming to a plateau could indicate that the maximum reward is nearing or that the reward structure requires tweaking. Upon observation during testing, it appears that the rewards are reaching the ceiling for the level.

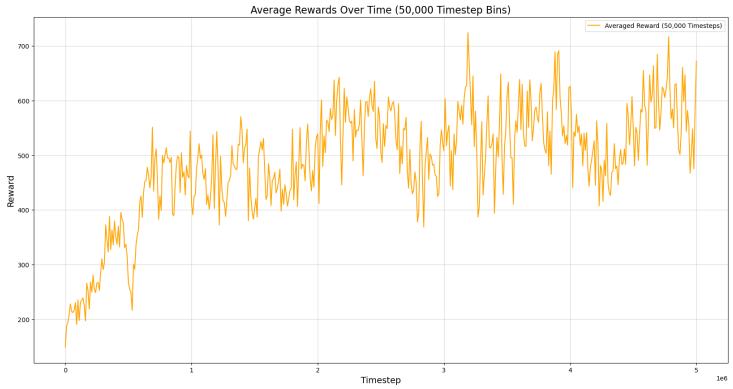



**Figure 5.5. Graph of rewards averaged over 50,000 timesteps during training**

### 5.1.4 Behavioral Analysis

The performance of the models was observed on three levels: 1-1, 6-1, and 7-1. These maps were chosen on the basis that the performance of both models is similar.

We noted that the FSM agent performs rather predictably in the sense that we could roughly gauge what actions it would take and where it would die. The RL agent performs stochastically due to its training by exploration. Due to this, we were unable to correctly guess the agent's actions at certain parts of the level. Considering that there is no randomized seed to the environment, it shows that the RL agent was not trained enough to perform predictably.

### 5.1.5 Implementation Effort

The current state of Gym-Super-Mario-Bros allows us access to low-level game variables like the number of coins collected and Mario's X position. Access to these variables speeds up implementation and eases efforts. In the event it doesn't

provide us access to these variables, computer vision techniques can be used to get some, but not all variables.

For the RL agent, the most time-consuming task is training and hyperparameter tuning. These factors are limited by available computational resources. Due to this, RL implementation took significantly more time than the FSM, but it produced a stabler and more adaptable model.

## 5.2 DOOM Evaluation

Evaluation for these agents is split into 3 maps with each map testing a unique aspect:

- **#1. Defend_the_center**: The map agents are developed for. Here, we evaluate agents' general performance.
- **#2. Basic**: A map with an unknown enemy not seen before by agents. It also presents a different environment. We evaluate agents' adaptability here.
- **#3. Defend_the_line**: A map combining aspects of the previous two maps featuring both seen and unseen enemies, and a new environment. Here, we focus on agents' overall adaptability and performance.

### 5.2.1 Completion Rate

For this metric, map 3 will not be considered as there is no set maximum score. On map 1, both agents didn't manage to achieve a perfect score. However, the RL agent prevails. On map 2, the FSM agent proves to be better as managed to get a perfect score each episode, proving its consistency. Although almost perfect, the RL agent can be seen struggling with some edge cases that happen on the map.

Finite State Machine

| map | avg killcount | max killcount |
|---|---|---|
| basic (max 1 kill) | 1.0 | 1.0 |
| defend_the_center (max 26 kills) | 13.0 | 23.0 |
| defend_the_line (unlimited kills) | 22.8 | 32.0 |

Reinforcement Learning - PPO

| map | avg killcount | max killcount |
|---|---|---|
| basic (max 1 kill) | 0.8 | 1 |
| defend_the_center (max 26 kills) | 21.1 | 25.0 |
| defend_the_line (unlimited kills) | 18.8 | 26.0 |

**Figure 5.6. A table of agents' kill counts**

### 5.2.2 Progression Rate

The RL agent shows a better average and overall performance. The huge gap between the FSM agent's average and best performance shows that the agent couldn't consistently achieve high progression rates on the map 1, as shown by the standard deviation. However, due to the simplistic nature of map 2, it consistently gets a full score whereas the RL agent achieves a full score 80% of the time. This could be due to the RL agent not having seen the enemy at all during training.

Finite State Machine

| map | avg killcount | max killcount | avg completion | best completion | standard deviation |
|---|---|---|---|---|---|
| basic (max 1 kill) | 1.0 | 1.0 | 100% | 100% | 0 |
| defend_the_center (max 26 kills) | 13.0 | 23.0 | 50% | 88.46% | 6.12 |
| defend_the_line (unlimited kills) | 22.8 | 32.0 | NA | NA | 5.25 |

Reinforcement Learning - PPO

| map | avg killcount | max killcount | avg completion | best completion | standard deviation |
|---|---|---|---|---|---|
| basic (max 1 kill) | 0.8 | 1 | 80% | 100% | 0.4 |
| defend_the_center (max 26 kills) | 21.1 | 25.0 | 81.15% | 96.15% | 3.12 |
| defend_the_line (unlimited kills) | 18.8 | 26.0 | NA | NA | 6.24 |

**Figure 5.7. A table of performance values for both agents**

A table (figure 5.8) is laid out to compare the average and standard deviation values for further analysis.

| Map | FSM | RL |
|---|---|---|
| basic | High average<br>Low standard deviation | High average<br>High standard deviation |
| defend_the_center | Low average<br>High standard deviation | High average<br>Low standard deviation |
| defend_the_line | High average (compared to performance of both agents)<br>Moderate standard deviation | Low average (compared to performance of both agents)<br>Moderate standard deviation |

**Figure 5.8. A table comparing the average and standard deviation of both agents**

### 5.2.3 Rewards Accumulation

We have converted the FSM agent's reward values to follow the same reward structure the RL agent uses as they use different reward structures. Figure 5.9 reflects this.

The reward values give us a better insight towards the overall performance of the agents as it accounts for mistakes made by agents. Here we can confirm that the RL agent has a better, and more stable performance across maps. Its subpar performance in the "basic" map could be attributed to lack of training as the environment is different and the shape of the enemy is much different compared to the other maps.

Comparing this with figure 5.7, we see that the FSM can perform well metric-wise, but its speed sabotages its performance in more dynamic environments. This is especially obvious for the "defend_the_center" map as it requires agents to defend itself while enemies surround them in all directions while requiring the agent to have quick and precise aim to be able to shoot enemies at varying distances.

Rewards

| map | FSM Avg | FSM Best | RL Avg | RL Best |
|---|---|---|---|---|
| basic (max 1 kill) | 77.6 | 95.0 | -20.3 | 95.0 |
| defend_the_center (max 26 kills) | 78 | 243.0 | 223.2 | 271.0 |
| defend_the_line (unlimited kills) | 219 | 329 | 203.8 | 285.0 |

**Figure 5.9. A table of reward values for both agents**

### 5.2.4 Behavioral Analysis

The FSM agent performs slower than the RL agent. The time-to-kill for each enemy is significantly higher than that of the RL model. This could be due to the inconsistent enemy detection. On top of that, it does not accurately detect moving enemies. Contrastingly, RL agent appears to be more adaptable on maps 2 and 3. This could be due to the similarities between the environments.

The FSM agent's actions are still predictable however, it stutters a lot, Surprisingly, the RL agent's actions were predictable, almost as well as predicting the FSM agent's performance.

### 5.2.5 Implementation Effort

The ability to access low-level game variables hastens its development. In the event these variables are inaccessible, the screen vides information on some, but not all, variables.

Processing the screen for vision gets tricky due to wall textures. Much time was spent experimenting with different vision approaches like the one mentioned in section 4.2.1.1. RL agent's vision was simpler to implement due to the reliability CNNs. Experimentation was required to find the right number of layers and select parameters but for the most part, the CNN policy from stable baselines 3 proved to work well.

Timewise, the RL agent's implementation took the longest due to hyperparameter tuning where it took six major versions of experimentation, each coupled with several iterations of smaller changes.

## 5.3 Hybrid Agent Evaluation

The implementation of the proposed hybrid agent doesn't follow the same evaluation structure as it is a proof-of-concept work. Here, we evaluate its pros and cons with regards to development, and potential issues that may arise.

### 5.3.1 Adaptability

At its current implemented state, it won't be able to adapt to even the slightest change of environments as the models were trained specifically for the selected map. However, this concept is promising provided it is designed well. The agent may perform well only if it was designed for all possible mechanics available in a selected video game. This being well designed states and their respective RL models.

### 5.3.2 Implementation Effort

The implementation duration is equivalent to that of the FSM and RL implementations collectively. However, more time was taken in the planning phase as the agent's architecture is more complex. Due to ViZDoom's customizability, we were able to plan and implement training easily. In the event a game doesn't offer a similar degree of customizability, we predict that this agent may not work well and thus, only those who have access to the game's source

code would be able to implement this hybrid agent successfully. Despite the brief training the RL models received, the hybrid agent performed relatively well and shows its promise.

(1485 words)

# CHAPTER 6: CONCLUSION

This project aimed to compare and analyze the performance, adaptability, and user experience of Finite State Machines (FSM) and Reinforcement Learning (RL) with the Proximity Policy Optimization (PPO) algorithm. The research question steers the project in the direction of understanding these approaches through the context of game-playing, by exploring these approaches through dynamic video game environments. These approaches were fared in both 2D and 3D games, namely Super Mario Bros. and Doom. Additionally, the final aim of the project drives us to find potential synergies between these approaches, and how a hybrid approach could effectively merge their strengths.

## 6.1 Summary of Methodologies

The FSM agent focuses on structured, rule-driven behavior, enabling a stable and predictable approach. It made use of standard computer vision techniques such as, grayscaling, thresholding, gaussian blur, edge detection, and contours.

In contrast, the RL agent focuses on learning optimal strategies by interacting with the game environment. It attempts to balance exploration and exploitation by a mixture of trying random moves and using previously found successful moves. It makes use of a Convolutional Neural Network (CNN) to detect important details of the environment.

Combining the strengths of both approaches, the hybrid agent aims at providing a stable, predictable, and adaptable approach. It makes use of FSMs for high-level decision making while using RL for low-level actions. The RL models were trained on scenarios specific to its paired state.

## 6.2 Evaluation and Key Findings

Quantitative metrics, such as completion rates, progression rates, and reward accumulation, revealed that the FSM agent performs consistently. However, it struggled with increasing game complexity. Qualitative metrics, such as behavioral analysis and implementation effort, revealed that its performance depended heavily on the computer vision technique used. It performs consistently but occasionally stutters when more textures are present.

The RL agent showed better learning capability and adaptability with increasing game complexity. This comes at a high cost of computational resources. Its quantitative metrics proved it can adapt in unfamiliar scenarios, but its qualitative metrics showed that it was heavily limited by the availability of computational resources. Additionally, despite having better computational power and resources, implementation time is higher due to factors like reward shaping and hyperparameter tuning.

The hybrid model showed some promise in balancing the structured approach of FSMs with the adaptability and performance of RL even though it's a proof-of-concept prototype. However, it required a higher amount of time and effort to implement.

## 6.3 Strengths and Limitations

FSMs, in this case, proved to be reliable and easier to implement and interpret. It is ideal for environments where rule-following is a critical core component of the game. Its source code is easy to follow and error handling is simple due to its modular approach. However, it isn't easily scalable and isn't adaptable to increasingly complex environments

RL can adapt to dynamic environments and optimize performance through continuous learning, or even curriculum learning. Its implementation is straightforward, but it is heavily dependent on its reward structure. On top of that, its implementation duration is dependent on available computational resources as they are directly proportional. Lastly, hyperparameter tuning requires attention to detail and patience, since it takes up a lot of time.

## 6.4 Implications

FSMs are perfect for game developers who want to implement enemies or other non-player characters in a video game. This is because they have access to low-level variables like enemy and player positions that can ease development efforts. Also, due to the availability of these variables, they may not need to rely on computer vision techniques to develop these agents. This could in return produce a stable and consistent agent. The modular nature of FSMs also provides better code readability.

RL models could be exploited by game developers as well as end users who do not have access to low-level game variables. These users include, but are not limited to, video game modders, gamers who like to experiment with video games, and researchers. Although, these variables can help improve the model drastically, it isn't needed for the most part. A very important factor to take note of is that these users must have access to decent computer hardware, like a mid-high range of Central Processing Units (CPU) and Graphical Processing Units (GPU). Otherwise, their efforts may be wasted.

These users could create AI bots or other non-player characters whereas game developers could use this model to implement highly skilled enemies, adding more depth to their video games. Researchers, on the other hand, can use the data in this report to find improved variations of RL architectures.

## 6.5 Recommendation for Future Work

Future research could focus on broadening the testing of game environments. It could explore various other game genres and more complex scenarios. This would provide deeper insights into the robustness of both approaches.

Another area of future work could focus on investigating other ways of implementing hybrid agents and potentially implementing a version opposite of our version: with RL focusing on high-level decisions and FSMs focusing on low-level actions. This would give insights into its performance, scalability, and stability. This could in return inspire other forms of architectures with regards to game-playing AIs that can rival popular methods like Deep Q-Networks (DQN).

Finally, improving the current state of the hybrid agent could be another area of future work. It would focus on the architecture proposed in this project and find ways to improve it further by exploring various ways to implement the FSM and RL models.

(905 words)

# CHAPTER 7: REFERENCES

[1] R. Liu, Nitish Sontakke, and S. Ha, “PM-FSM: Policies Modulating Finite State Machine for Robust Quadrupedal Locomotion,” 2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), vol. 9, pp. 4063–4069, Oct. 2022, doi: https://doi.org/10.1109/iros47612.2022.9982259.

[2] S. Hwang, K. Lee, H. Jeon, and D. Kum, “Autonomous Vehicle Cut-In Algorithm for Lane-Merging Scenarios via Policy-Based Reinforcement Learning Nested Within Finite-State Machine,” vol. 23, no. 10, pp. 17594–17606, Jan. 2022, doi: https://doi.org/10.1109/tits.2022.3153848.

[3] C. Wang, X. Chen, Z. Yu, Y. Dong, K. Chen, and P. Gergondet, “Robust humanoid robot vehicle ingress with a finite state machine integrated with deep reinforcement learning,” International Journal of Machine Learning and Cybernetics, Oct. 2024, doi: https://doi.org/10.1007/s13042-024-02407-w.

[4] Uraz Cengiz Türker, R. M. Hierons, Khaled El-Fakih, M. R. Mousavi, and I. Y. Tyukin, “Accelerating Finite State Machine-Based Testing Using Reinforcement Learning,” IEEE Transactions on Software Engineering, vol. 50, no. 3, pp. 574–597, Jan. 2024, doi: https://doi.org/10.1109/tse.2024.3358416.