SAPIENZA
UNIVERSITÀ DI ROMA

Pawnimation - Cinematic Chess Visualization in Unreal Engine 5 with Reinforcement Learning Camera Control

Facoltà di Ingegneria dell'Informazione, Informatica e Statistica

Corso di Laurea Magistrale in Engineering in Computer Science

Candidate

Simone Di Cocco

ID number 1934818

Thesis Advisor

Prof. Paolo Russo

Academic Year 2024/2025

Thesis defended on 24 October 2025
in front of a Board of Examiners composed by:

Prof. Aristidis Anagnostopoulos (chairman)
Prof. Luca Becchetti
Prof. Giorgio Grisetti
Prof. Paolo Russo
Prof. Roberto Navigli
Prof. Fabio Patrizi
Prof. Gabriele Proietti Mattia

Pawnimation - Cinematic Chess Visualization in Unreal Engine 5 with Reinforcement Learning Camera Control
Master's thesis. Sapienza – University of Rome

This thesis has been typeset by LaTeX and the Sapthesis class.

Author's email: dicocco1934818@studenti.uinroma1.it

A mio nonno Nicola,
che sono certo vegli sempre su di me.
- Il tuo Simon

Ringraziamenti

È faticoso, ma anche doveroso e tutto sommato gradito, spendere qualche parola per le persone che hanno avuto la fortuna di impiegare gran parte del loro tempo beandosi della mia compagnia.

Desidero quindi ringraziare:

i miei genitori, per l'impegno e la dedizione profusi nel crescermi - con risultati straordinari - e per il supporto fornitomi in ogni modo possibile;
Sara, per ricordarmi quotidianamente che, se riesco a sopportare la convivenza con lei, allora non esistono ostacoli insormontabili nella vita;
Ilaria, che, nonostante le origini pastorali, con la sua dolcezza è sempre in grado di farmi sciogliere come neve al sole e di tranquillizzarmi quando è necessario;
Alessio, che, con ~~la sua sconfinata generos~~ ~~la sua illimitata disponib~~ ~~il suo costante buonum~~ ~~la sua cccezionale freddez~~ tutti i suoi difetti, rimane ~~la persona migliore nel prepararmi per gli esam~~ il migliore amico che possa desiderare;
Andrea, che, con la sua riverberante, ruggente e irripetibile erre e la sua irriducibile e straordinaria personalità, ha arricchito rimarchevolmente entrambi i nostri percorsi universitari (triennale e magistrale), e spero profondamente proseguirà ad arricchire anche le prossime frizzanti avventure che affronteremo;
i miei nonni, sempre in grado di strapparmi una risata con i loro caratteri strampalati, ma anche capaci di donare affetto come solo i nonni sanno fare;
i miei parenti (persino Davide), fortunamente meno serpenti di quanto il proverbio suggerisca, con cui ho trascorso - e trascorro tuttora - un numero incalcolabile di serate di risate e prese in giro;
i miei amici e colleghi - liceali e universitari - per aver arricchito e alleggerito il mio percorso di studi nel corso di questi anni, che senza di loro sarebbero stati molto meno divertenti;
ed infine Cocco, che col suo tanfo letale e il suo carattere schizofrenico è ormai da quattro anni parte della nostra famiglia, che non potrebbe esserne più felice.

Abstract

Pawnimation is a project developed in Unreal Engine 5 that transforms the playback of chess matches into a cinematic and visually compelling experience. By parsing PGN (Portable Game Notation) files, the system reconstructs historical games and replays them in a fully rendered 3D environment enriched with music and a space-themed background. A distinctive feature of the implementation is the destruction of captured pieces: whenever a piece is taken, its static mesh is dynamically replaced by a pre-fractured geometry that shatters into debris, producing a visually striking effect.

Beyond the implementation of PGN parsing and real-time replay logic, the experimental contribution of this work is the introduction of a camera controlled by Reinforcement Learning (RL). Unlike conventional camera systems, which are typically fixed or driven by simple heuristics, the RL agent was trained to select cinematographic viewpoints that enhance the visual impact of each move. Positioned at board level, as if an observer were standing on a chessboard of human-sized pieces, the agent learnt to balance functional visibility with aesthetic composition. Once a satisfactory viewpoint is reached, the agent remains fixed until the next move, ensuring continuity of the shot.

The originality of this research lies in applying Reinforcement Learning not to navigation or object tracking, but to the pursuit of cinematic framing. *Pawnimation* therefore provides both a technical demonstration of chess visualization and a novel exploration of AI-driven cinematography in interactive environments.

Contents

List of Figures

Chapter 1

Introduction

This chapter introduces the context, motivation, and objectives of the thesis, outlining both its technical foundations and experimental contributions. It also presents the structure of the work and highlights the originality of the proposed approach.

1.1 Context and Motivation

Chess has long been regarded not only as one of the most iconic strategy games in human history, but also as a cultural symbol of logic, intelligence, and competition. While the notational systems such as PGN (Portable Game Notation) enable games to be stored, transmitted, and replayed [13, 16], their traditional representation is purely textual and abstract. Even modern digital chess platforms typically focus on functionality and clarity, offering a two-dimensional board with minimal graphical embellishments.

At the same time, advances in real-time graphics engines such as Unreal Engine 5 have opened the door to increasingly immersive and cinematic experiences. In parallel, the rise of interactive visualization has created opportunities to reinterpret traditional domains - including board games - through techniques borrowed from video games, film production, and simulation [3, 6]. These technologies are no longer limited to entertainment, but are increasingly explored in research and educational contexts to improve engagement, accessibility, and the perception of complex information.

Within this landscape, the idea of transforming the replay of chess games into a visually compelling and cinematic event represents a natural yet unexplored extension. Instead of presenting a match as a simple sequence of moves on a flat board, it can be reimagined as a spectacle that combines intelligent camera work and striking visual effects [1, 4]. This perspective not only enhances the aesthetic appreciation of the game, but also provides a new form of narrative that emphasizes the tension and drama inherent in each move.

1.2 Objectives of the Project

The project presented in this thesis, called *Pawnimation*, aims to reinterpret the playback of chess matches through cinematic techniques and advanced visual effects. Rather than limiting the game to its traditional abstract representation, the system seeks to create an engaging and immersive experience that combines the rigor of chess with the aesthetics of interactive graphics. To achieve this goal, the work is structured around two complementary components: a technical foundation and an experimental contribution.

The technical foundation focuses on the faithful reproduction of chess games starting from PGN files. This involves parsing the PGN notation, reconstructing the sequence of moves, and implementing a replay system inside Unreal Engine 5. In addition, visual effects are introduced to enhance the impact of captures: instead of simply removing a piece from the board, the captured piece is replaced by a pre-fractured geometry that breaks apart into fragments, producing a dramatic destruction effect.

The experimental contribution concerns the design of a camera system guided by Reinforcement Learning (RL) [9, 10, 2, 11, 15, 7]. Unlike conventional approaches, where camera positions are either fixed or determined by simple rules, the RL agent was trained to select viewpoints that balance visibility with cinematic quality. The agent operates at board level, as if the pieces were human-sized and the camera an observer standing among them. By optimizing a reward function that emphasizes the framing of the move, the agent is able to generate dynamic and aesthetically pleasing perspectives [12, 18, 19, 5, 17, 8].

Taken together, these two components define the objectives of the thesis: on the one hand, to implement a technically solid replay system with enhanced visual fidelity; on the other hand, to explore an original application of RL for AI-driven cinematography, demonstrating the potential of combining game technology with machine learning to achieve novel forms of interactive visualization.

1.3 Contributions of the Thesis

The work presented in this thesis makes several contributions that span both the technical implementation of the system and the experimental exploration of AI-driven cinematography. The main contributions can be summarized as follows:

- **Implementation of a PGN-based replay system:** *Pawnimation* develops a pipeline that parses PGN (Portable Game Notation) files and reconstructs chess matches inside Unreal Engine 5, enabling the faithful playback of historical games [13].
- **Dynamic visualization of captures:** captured pieces are not simply removed from the board, but are replaced by pre-fractured geometries that break apart into fragments. This introduces a novel destruction effect that adds cinematic intensity to the replay.
- **Integration of audiovisual design:** the environment is enriched with a

space-themed setting and accompanied by music, reinforcing the immersive and dramatic qualities of the experience.

- **Design of a Reinforcement Learning camera system:** the thesis introduces an RL-based agent trained to select cinematographic viewpoints that emphasize both visibility and aesthetic composition, positioning the camera at board level as if the pieces were human-sized [12, 18, 19].
- **Definition of a tailored reward function:** a set of incentives and penalties was designed to help the RL agent capture each move in an attractive way, making sure the key parts of the action stay in view and well-centered.
- **Exploration of a novel application domain:** this work applies Reinforcement Learning to the problem of cinematic camera control, contributing to the broader discussion on AI-driven cinematography in interactive environments [3, 8, 17].

Together, these contributions establish *Pawnimation* both as a functional system for chess replay and as a research experiment that bridges computer graphics, artificial intelligence, and interactive visualization.

1.4 Structure of the Thesis

The remainder of this thesis is organized as follows:

- **Chapter 2 - Background and Related Work:** reviews the existing literature and technologies relevant to the project, including cinematic visualization in chess games and the use of Reinforcement Learning in camera control.
- **Chapter 3 - Theory:** introduces the theoretical foundations underlying the project, with particular attention to the principles of Reinforcement Learning and camera modeling.
- **Chapter 4 - Implementation:** describes the practical development of the system in Unreal Engine 5, covering the PGN replay pipeline, the destruction mechanics, and the design and training of the RL-based camera agent.
- **Chapter 5 - Experiments and Discussion:** presents the outcomes of the implementation, evaluates the behaviour of the RL camera, and discusses the strengths, limitations, and qualitative impact of the approach.
- **Chapter 6 - Conclusions and Future Work:** summarizes the main findings of the thesis, reflects on its contributions, and outlines possible directions for extending the work.

The following chapter now turns to the academic and technological background, situating this work within the broader context of related research and prior developments.

Chapter 2

Background and Related Work

2.1 Background

Virtual cinematography is the process of automatically or semi-automatically controlling cameras in digital environments to achieve visually expressive and coherent results. The field has evolved from simple camera placement heuristics to complex systems capable of simulating the work of human cinematographers in real-time 3D scenes. Early approaches relied on explicit rule-based systems and declarative camera control languages, where camera shots were generated according to predefined idioms or cinematographic conventions. These methods were particularly effective in reproducing traditional film grammar in virtual contexts, but they often lacked adaptability and stylistic diversity [1].

With the growing realism of interactive 3D environments, automated camera control has become a crucial component for enhancing narrative immersion and visual engagement. Research in this domain has explored geometric optimization, constraint satisfaction, and behaviour-based models for smooth camera motion and composition. More recent frameworks have started to bridge computer graphics and artificial intelligence, introducing mechanisms capable of dynamically adapting the viewpoint to contextual and narrative cues [6, 18, 3].

In parallel, Reinforcement Learning (RL) has emerged as a promising paradigm for decision-making in continuous and dynamic environments. By learning through trial and error, RL agents can autonomously discover control strategies that maximize a given reward signal. In the context of camera control, RL offers a compelling alternative to manually encoded rules: rather than being explicitly told how to compose a shot, the agent learns to select camera positions that maximize aesthetic, geometric, or narrative rewards [12, 19, 5, 17]. Such a formulation is particularly well suited for cinematic visualization, where a balance must be achieved between functional visibility and artistic expression.

2.2 Related Work

Automated cinematography has been widely investigated in both the film industry and academic research. Early contributions introduced declarative frameworks for virtual camera control, defining idioms for specific actions such as dialogues or

fights, and selecting camera shots according to pre-authored cinematographic recipes. These systems established a foundation for rule-based virtual direction, but they required extensive manual setup and were limited to discrete, predictable scenarios [1]. Later approaches introduced optimization techniques to compute camera trajectories that satisfy multiple cinematographic constraints simultaneously, allowing for more continuous and natural motion in dynamic 3D environments [18].

A significant conceptual shift occurred with the introduction of importance-driven and narrative-driven camera models. Instead of linking camera placement to low-level character actions, these systems compute the narrative importance of characters and events, and use it to drive shot composition, camera switching, and editing. The resulting cinematography better conveys the narrative hierarchy and emotional focus of the scene, providing replays that resemble human-directed sequences [6]. This line of work highlighted the potential of contextual information, such as character motivation, relevance, or emotion, to influence the choice of viewpoint and framing.

Building on these principles, more recent methods have explored data-driven or learning-based camera control. Neural models and generative frameworks can learn mappings between actor behaviour, spatial configuration, and desirable camera motion, sometimes incorporating aesthetic rules such as the rule of thirds or stylistic parameters derived from cinematic datasets [18, 8]. Such approaches aim not only to ensure visibility and coherence, but also to enhance immersion and emotional resonance. They often include perception-based evaluations and user studies to quantify the aesthetic or immersive quality of the generated sequences.

Other contributions have emphasized the integration of artificial intelligence in the broader filmmaking pipeline, envisioning AI systems capable of assisting or even automating script generation, shot composition, and editing [4, 3]. These perspectives frame AI not merely as a technical optimization tool, but as an emerging creative collaborator in the future of cinematography. Within this evolving landscape, Reinforcement Learning provides a particularly flexible framework for modeling cinematic decision-making. RL-based agents can adapt camera placement to unpredictable events and dynamically evolving scenes, making them suitable for both interactive applications and systems where aesthetic adaptability is essential [12, 19].

In virtual environments, the challenge lies in achieving coherence between camera autonomy and narrative intent. While optimization and rule-based systems ensure consistency and control, learning-based approaches such as RL introduce the ability to discover novel, human-like camera behaviours. The use of reward functions that encode geometric visibility, compositional balance, or narrative relevance allows these agents to generate shots that satisfy both technical and artistic criteria. This paradigm shift (from manually designed behaviours to data-driven or learned cinematography) represents a major step toward autonomous camera systems capable of aesthetic reasoning and adaptive storytelling [12, 18, 3].

Chapter 3

Theory

3.1 Chess Representation

3.1.1 Notation Systems

The notation system used to record chess games has evolved over several centuries, adapting to the needs of players, analysts, and digital processing. Early methods such as *descriptive notation* (e.g., `P-K4`) were dominant in English-speaking countries until the late twentieth century, but they suffered from ambiguity and lack of universality. This led to the adoption of the modern *algebraic notation*, which expresses moves in a coordinate-based format and is now the international standard [16, 13].

In algebraic notation, the chessboard columns (files) are labeled `a` to `h`, and the rows (ranks) from `1` to `8`. Each square can therefore be uniquely identified by its file and rank (for instance, `e4`). A move is expressed by indicating the destination square, with the piece type as a prefix if needed (e.g., `Nf3` denotes a knight moving to `f3`). Captures are denoted with an "x" symbol (`Bxe6`), and special notations are used for castling (`O-O`, `O-O-O`) and pawn promotion (`e8=Q`).

A variant called *long algebraic notation* (LAN) explicitly includes both origin and destination squares (e.g., `e2e4`), which is especially suitable for computer-based representations of games, where disambiguation and coordinate processing are essential.

Algebraic	Long Algebraic	Description
e4	e2e4	White pawn moves from e2 to e4
c5	c7c5	Black pawn moves from c7 to c5
Nxf2	Nh1xf2	Knight captures from h1 to f2

Table 3.1. Comparison between standard and long algebraic notation.

3.1.2 Game Recording Formats

Portable Game Notation (PGN)

The *Portable Game Notation* (PGN) is a widely adopted plain-text format for recording chess games. Introduced by Steven J. Edwards in the early 1990s, PGN

was designed to be both human-readable and machine-parseable, making it ideal for digital databases and software applications.

A PGN file consists of two main parts:

1. A header section, enclosed in square brackets, containing metadata tags such as event name, players, date, and result.
2. The move text section, listing the sequence of moves in algebraic notation.

```
[Event "Fischer Random blitz 2018"]
[Site "Baerum NOR"]
[Date "2018.02.13"]
[White "Magnus Carlsen"]
[Black "Hikaru Nakamura"]
[Result "1-0"]

1. a4 e6 2. a5 a6 3. e4 Nf6 4. e5 Nd5 5. Nf3 f6 6. exf6 gxf6 1-0
```

PGN supports additional annotations for comments, evaluations, and variations, making it a flexible format for both archival and analytical purposes. In *Pawnimation*, PGN parsing serves as the foundation for reconstructing and visualizing chess games as cinematic replays in Unreal Engine 5.

Forsyth–Edwards Notation (FEN)

While PGN describes entire games, the *Forsyth–Edwards Notation* (FEN) provides a compact representation of a single board state. It encodes all the necessary information to resume a game from a given position in a single line of text. The notation consists of six space-separated fields:

1. Piece placement (from rank 8 to 1)
2. Active color (“w” or “b”)
3. Castling availability
4. En passant target square
5. Halfmove clock
6. Fullmove number

```
r1b1k2r/p1pp1ppp/2p3n1/2b1P3/1qP5/3B2Q1/PP1N1PPP/R1B1K2R b KQkq - 8 12
```

This FEN string describes a mid-game position where both players have already developed most of their pieces. It can be read as follows:

- **r1b1k2r**: the 8th rank (Black’s back rank) – black rooks on a8 and h8, black bishop on c8, black king on e8.
- **p1pp1ppp**: the 7th rank – black pawns on a7, c7, d7, f7, g7, h7.

- **2p3n1**: the 6th rank – black pawn on c6, knight on g6.
- **2b1P3**: the 5th rank – black bishop on c5, white pawn on e5.
- **1qP5**: the 4th rank – black queen on b4, white pawn on c4.
- **3B2Q1**: the 3rd rank – white bishop on d3, white queen on g3.
- **PP1N1PPP**: the 2nd rank – white pawns on a2, b2, e2, f2, g2, h2; white knight on d2.
- **R1B1K2R**: the 1st rank – white rooks on a1 and h1, bishop on c1, king on e1.

The remaining fields specify additional information about the game state. The character `b` indicates that it is **Black's turn to move**, and the sequence `KQkq` shows that both sides still retain their full castling rights (King-side and Queen-side). The dash symbol (`-`) denotes that there is currently **no en passant target square**.

The value `8` corresponds to the **halfmove clock**, which counts the number of half-moves since the last pawn advance or capture. This counter is used for enforcing the *fifty-move rule*, which declares a draw if fifty consecutive moves occur without any pawn move or piece capture.

Finally, the last number, `12`, represents the **fullmove number**, which increments after each move by Black. In this position, it means that the game has reached the twelfth full move, i.e., twenty-three half-moves have been played and Black is about to make the twenty-fourth.

Figure 3.1. Board representation of the FEN position above.

FEN is particularly useful for debugging and intermediate analysis, as it allows the recreation of any position encountered during a PGN playback or simulation.

Other Notation and Exchange Formats

Other formats such as the *Standard Algebraic Notation* (SAN), *Universal Chess Interface* (UCI), or JSON-based schemas exist for different purposes, but they are less relevant to this work. Among all, PGN remains the most accessible and widely supported, serving as the backbone for digital chess databases and visualization systems.

3.2 Reinforcement Learning

Reinforcement Learning (RL) is a branch of machine learning concerned with sequential decision-making, where an agent interacts with an environment to maximize cumulative reward over time [9, 7]. Unlike supervised learning, RL does not require explicit input-output pairs but instead relies on trial-and-error interactions, receiving feedback through a reward signal. This framework has been successfully applied in a wide range of domains, from game playing and robotics to cinematography and autonomous camera control [12, 18, 19].

3.2.1 Fundamentals of Reinforcement Learning

The formalism of RL is commonly described using a *Markov Decision Process* (MDP), defined by the tuple $(\mathcal{S}, \mathcal{A}, P, R, \gamma)$, where $\mathcal{S}$ is the set of states, $\mathcal{A}$ is the set of actions, $P(s'|s,a)$ represents the state transition probability, $R(s,a)$ is the reward function, and $\gamma \in [0,1]$ is the discount factor [9]. The agent's behaviour is governed by a *policy* $\pi(a|s)$, which specifies the probability of taking action a in state s. The objective is to find an optimal policy π^* that maximizes the expected cumulative discounted reward, also called the *return*:

$$G_t = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}. \tag{3.1}$$

Two key functions are often introduced: the *state-value function* $V^\pi(s)$, representing the expected return from state s under policy π, and the *action-value function* $Q^\pi(s,a)$, representing the expected return from taking action a in state s and following policy π thereafter. These functions underpin most RL algorithms, whether value-based, policy-based, or actor-critic [9, 7].

Exploration-exploitation trade-offs are central to RL. The agent must balance exploring unknown actions to improve its knowledge of the environment against exploiting the actions it already knows yield high rewards. Various strategies exist, including ϵ-greedy policies, softmax action selection, and upper confidence bounds [9, 10].

3.2.2 Classical RL Algorithms

Early RL methods focused on tabular approaches, suitable for environments with discrete and small state-action spaces. One of the most foundational algorithms is *Q-learning*, introduced by Watkins [9]. Q-learning is an off-policy, value-based

method that iteratively updates the action-value function $Q(s, a)$ according to the Bellman equation:

$$Q(s,a) \leftarrow Q(s,a) + \alpha \left[R_{t+1} + \gamma \max_{a'} Q(s',a') - Q(s,a) \right], \tag{3.2}$$

where α is the learning rate, R_{t+1} is the reward received after taking action a in state s, γ is the discount factor, and s' is the next state. Intuitively, this update moves the current estimate $Q(s, a)$ toward a target composed of the immediate reward plus the discounted value of the best action in the next state. Through repeated interactions and sufficient exploration, $Q(s, a)$ converges to the optimal action-value function Q^*, which can be used to derive the optimal policy by selecting, at each state, the action with the highest Q value [9]. Tabular methods such as Q-learning and SARSA laid the foundation for understanding the principles of temporal difference learning, bootstrapping, and policy evaluation [7].

3.2.3 Deep Reinforcement Learning

The advent of Deep Reinforcement Learning (Deep RL) combined RL with deep neural networks, allowing agents to operate in high-dimensional, continuous, and partially observable environments [10, 11, 2]. Deep RL leverages neural networks as function approximators for value functions, policies, or both, extending the applicability of classical RL algorithms. For example, the Deep Q-Network (DQN) uses a convolutional neural network to approximate $Q(s, a)$ from raw pixel inputs, achieving human-level performance on Atari 2600 games [10, 2].

Deep RL algorithms are often categorized into three families:

1. **Value-based methods**: estimate the expected return for each action, e.g., DQN [10].
2. **Policy-based methods**: directly parameterize the policy and optimize the expected return using gradient ascent.
3. **Actor-Critic methods**: combine value function estimation (critic) with policy optimization (actor), benefiting from lower variance and improved learning stability [2].

3.2.4 Policy Gradient Methods and Proximal Policy Optimization

Policy-based methods optimize a parameterized policy $\pi_\theta(a|s)$ by directly maximizing the expected return:

$$J(\theta) = \mathbb{E}_{\pi_\theta}[G_t]. \tag{3.3}$$

where G_t is the cumulative discounted reward. The *policy gradient theorem* provides a formal expression for the gradient of the expected return with respect to the policy parameters θ:

$$\nabla_\theta J(\theta) = \mathbb{E}_{s\sim d^\pi, a\sim\pi_\theta} \left[\nabla_\theta \log \pi_\theta(a|s) Q^\pi(s,a) \right], \tag{3.4}$$

where $d^\pi(s)$ is the discounted state visitation distribution under policy π, and $Q^\pi(s,a)$ is the action-value function under policy π. This gradient can be interpreted as adjusting the policy parameters in the direction that increases the probability of actions that yield higher returns. In practice, variance reduction techniques, such as using an advantage function instead of the raw $Q^\pi(s,a)$, are commonly applied to stabilize learning [9, 7].

Proximal Policy Optimization (PPO) is a modern, on-policy policy gradient algorithm designed for stable and efficient learning in high-dimensional action spaces [12, 15]. PPO uses a surrogate objective function with a clipping term to prevent large, destabilizing updates to the policy:

$$L^{CLIP}(\theta) = \mathbb{E}_t \left[\min \left(r_t(\theta)\hat{A}_t, \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t \right) \right], \tag{3.5}$$

where $r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t)}$ is the probability ratio between the new and old policies, $\hat{A}_t$ is an estimate of the advantage function, and ϵ is a small hyperparameter controlling the clipping range. The clipping ensures that if the policy update would change the probability of an action too much, the objective is limited, avoiding overly aggressive updates. Intuitively, PPO encourages improvements in the policy while maintaining stability and preventing catastrophic drops in performance, making it particularly suitable for continuous control tasks such as camera movement in cinematic applications [12, 15].

3.2.5 Comparison with Value-Based Methods

Although policy-based approaches like PPO optimize the policy directly, value-based methods such as Q-learning estimate the action-value function and derive a policy indirectly by selecting the action with the highest Q value. Each approach has strengths and limitations:

- **Value-based methods** are often simpler and have strong theoretical guarantees in discrete environments [9], but they struggle with high-dimensional or continuous action spaces.
- **Policy-based methods** can naturally handle continuous actions and stochastic policies, but can suffer from high variance in gradient estimates.
- **Actor-Critic methods**, including PPO, combine the advantages of both approaches by learning a value function to reduce variance while directly optimizing the policy [2, 15].

This comparison motivates the choice of PPO for applications requiring continuous camera control, where stability and smooth policy updates are essential [12, 18].

3.2.6 Applications in Cinematography and Camera Control

RL has been increasingly applied to automatic camera control, both in virtual cinematography and interactive environments. Early approaches explored cinematic

camera placement and narrative-driven camera movement [1, 6]. Recent work extends these ideas using Deep RL and PPO to optimize camera trajectories for immersive experiences [12, 18, 19, 5, 8].

For instance, Passalis and Tefas [12] applied deep RL to control a frontal person close-up camera, demonstrating how actor-critic methods can produce visually appealing shots. Wu et al. [18] introduced enhanced immersion through automatic camera trajectory optimization, while Yang et al. [19] and Fang et al. [5] extended RL to multi-agent PTZ camera setups, optimizing both framing and visual enhancement. These studies illustrate the potential of RL, and specifically PPO, for cinematographic tasks requiring fine-grained, continuous control.

3.2.7 Summary

In summary, Reinforcement Learning provides a principled framework for sequential decision-making under uncertainty. Classical methods, such as Q-learning and SARSA, form the theoretical foundation, while Deep RL enables applications in high-dimensional spaces. Policy gradient methods, particularly PPO, offer stable learning for continuous control tasks, making them ideal for applications in automatic camera control and cinematic visualization. The choice between value-based and policy-based methods depends on the environment's characteristics, action space, and desired policy behaviour. RL-based approaches have already shown significant success in cinematography, demonstrating the synergy between AI and visual storytelling [12, 18, 19, 5].

Chapter 4

Implementation

The implementation of the *Pawnimation* project is based on the Unreal Engine 5 software and focuses on transforming the textual representation of a chess match into a fully animated, cinematic visualization. The system receives as input a Portable Game Notation (PGN) file describing the entire sequence of moves played in a real chess game. Unlike traditional chess engines, the purpose of this implementation is not to evaluate move legality or compute optimal plays, but to reproduce and visualize existing matches through dynamic animations, cinematic lighting, and camera movements driven by reinforcement learning.

At a high level, the implementation is composed of three main subsystems:

1. **PGN Parsing and Move Representation** - The PGN text is parsed directly in Unreal Engine using Blueprints and regular expressions. Each move is converted into a structured data record (`Move`), storing information such as the moving piece, source and destination cells, and capture, castling, promotion, and check flags.

2. **Cinematic Playback and Effects** - The parsed PGN sequence is replayed on a 3D chessboard where each piece is represented by an actor capable of movement, animation, and destruction. The system supports dynamic motion styles and cinematic sequences for the game's beginning and end, integrates physically simulated captures, and includes synchronized audio and background visuals. Lighting and environment are designed to enhance the cinematic atmosphere, providing a cohesive visual and auditory experience.

3. **Camera Control via Reinforcement Learning** - A reinforcement learning (RL) agent controls a movable camera around the chessboard. Before each move, the agent selects a camera position and orientation that maximize a reward function designed to promote aesthetically satisfying shots. Rewards are based on criteria such as the spatial proximity to the move's midpoint, the centrality of the action in the camera's frustum, and the visibility of both starting and ending cells. The RL agent operates in cooperation with four static cinematic cameras that offer alternative viewpoints to enhance dynamism and variability (Figure 4.1).

The interaction between these modules is orchestrated through a Blueprint-based

Figure 4.1. Top-down view of the chessboard from one of the static cinematic cameras.

logic implemented in the `Level Blueprint`. Upon starting the level, the system performs a first invisible playback of the entire match, during which all animations have zero duration. This preliminary run generates a series of *snapshots* of the chessboard - one for each move - representing the board configuration after that move. These snapshots are later reused for several purposes, including instantaneous navigation through the timeline, move validation, and environment switching during RL-based camera training.

Through this architecture, *Pawnimation* integrates procedural animation, cinematic presentation, and artificial intelligence into a coherent system that automatically transforms a chess game into a visually immersive film-like experience.

4.1 PGN Parsing and Move Representation

The first core subsystem of *Pawnimation* is the one responsible for interpreting the Portable Game Notation (PGN) input and translating it into a structured set of executable actions inside Unreal Engine. This stage represents the conceptual bridge between symbolic chess notation and a physically simulated, cinematic environment. All the logic described in this section has been implemented entirely in Unreal Engine Blueprints and operates inside the `Level Blueprint` of the main scene.

4.1.1 PGN Interpretation and Regular Expression Parsing

The input PGN string is retrieved at the beginning of the simulation from the input field located in the main menu (Figure 4.2). This interface allows the user to paste or type a complete PGN record before launching the cinematic replay.

The parsing is performed by the function `Parse Input`, which uses regular expressions to decompose the PGN text into its elementary syntactic components.

Figure 4.2. Main menu interface of *Pawnimation*, featuring the PGN input field and control options for starting the simulation.

A dedicated regular expression pattern is defined in the Blueprint variable `Pattern`, designed to capture all relevant move features within a single expression. This pattern supports the following components:

- Identification of the moving piece (e.g., `N`, `B`, `R`, `Q`, `K`); if omitted, a pawn move is assumed.
- Optional specification of the originating file and rank (for move disambiguation).
- The capture indicator (`x`), used to flag taking moves.
- Destination square, expressed in algebraic notation (e.g., `e4`, `h7`).
- Promotion clause (`=Q`, `=R`, `=B`, `=N`).
- Castling moves (`O-O`, `O-O-O`).
- Check (`+`) and checkmate (`#`) indicators.

For each regex match, the captured tokens are used to instantiate a structured object of type `Move`. This object is an Unreal `UserDefinedStruct` whose fields explicitly encode all the semantic elements of a chess move:

- `Piece` (type of moving piece)
- `WhitePiece?` (boolean flag indicating color)
- `FromFile`, `FromRank` (origin coordinates, optional)
- `ToFile`, `ToRank` (destination coordinates)
- `Capture?` (boolean)

- `Promotion?`, `PromotedPiece`
- `IsCastling?`, `IsQueensideCastling?`
- `Check?`, `Checkmate?`

The resulting list of `Move` structures represents a complete, machine-readable encoding of the PGN. Each element corresponds to a frame of the future cinematic sequence, and all subsequent gameplay logic operates exclusively on this representation.

4.1.2 Move Disambiguation and Piece Detection

After parsing, each move must be mapped to the actual piece actor present on the 3D chessboard. This is a non-trivial operation, especially when several identical pieces could reach the same destination (for instance, two knights both capable of moving to the same square).

The detection algorithm, implemented in Blueprint under the function `Detect Piece`, performs the following steps:

1. It identifies all board actors matching the desired piece type and color.
2. For each candidate, it computes the potential path from the piece's current position to the target square.
3. It temporarily applies that move to a snapshot of the board (see Section 4.1.3) to test whether the resulting configuration would leave the King in check.
4. It invokes the validation function `Would King Be In Check?` using this snapshot as input. If the King is under attack in the simulated configuration, that candidate piece is excluded from consideration.
5. The algorithm returns the first valid candidate that produces a check-free position, ensuring that the move is consistent with real chess constraints.

Even though *Pawnimation* does not function as a chess engine in the traditional sense (since it does not verify global legality or generate new moves) the inclusion of this local legality test guarantees internal consistency and prevents the system from displaying logically impossible animations.

4.1.3 Snapshot Generation and Board State Tracking

A central architectural component of the system is the snapshot mechanism, which serves both as a data cache and a validation tool. At level startup, inside the `BeginPlay` event, the system performs an invisible first run of the entire chess match. During this preliminary simulation, all movement animations are assigned a duration of zero seconds, making the process imperceptible to the viewer.

The goal of this pass is to precompute a complete series of *board snapshots* - one for each move in the PGN sequence. Each snapshot contains a full description of the

board state after the corresponding move has been executed, including the positions and types of all pieces.

The stored snapshots are later used for three distinct purposes:

1. **Instantaneous navigation:** they allow the system to jump to any move in the match timeline without replaying previous moves, a feature essential for debugging, replay scrubbing, and camera testing.

2. **Move validation:** they provide the contextual input for the `Would King Be In Check?` function, which analyzes hypothetical board states to determine whether the King would be under threat.

3. **Environment flexibility for reinforcement learning:** each snapshot can be used to instantly instantiate a different environment configuration, allowing the RL camera agent to explore multiple scenarios efficiently.

This dual nature - simultaneously logical and cinematic - makes the snapshot subsystem a cornerstone of the implementation. It ensures that every rendered frame corresponds to a coherent game state, preserving both visual and structural consistency across the project.

4.1.4 Move Execution Logic

Once the piece corresponding to a parsed move has been identified, the `Play Move` function is responsible for animating the movement and updating the board state. The function receives a single `Move` structure and the duration of the animation as input and renders the corresponding action on screen.

The internal logic follows these steps:

1. The system breaks down the `Move` struct and extracts all necessary parameters (piece type, target coordinates, capture status, and special move flags).

2. The appropriate piece actor is commanded to execute its movement through a call to `Move To Cell`, which triggers the translation and animation sequence.

3. If the move involves a capture, the victim piece's mesh is replaced with a pre-fractured Geometry Collection, which is then physically simulated under the Chaos engine to produce a realistic shattering effect (Figure 4.3).

4. The system handles special cases: **en passant**, **castling**, and **promotion**. Each of these is implemented as a branch within the Blueprint logic, updating both visual and logical representations accordingly.

5. After execution, a new snapshot of the board is created and stored (only during the first invisible run).

(a) Initial impact. **(b)** Fragmentation begins. **(c)** Debris dispersal.

Figure 4.3. Sequence showing the White Queen capturing the Black Queen, whose mesh shatters into fragments through Chaos physics simulation.

4.1.5 Special Moves and Exception Handling

Each class of special chess move is treated with custom Blueprint logic:

Castling. When the `IsCastling?` flag is true, both the King and the corresponding Rook are moved simultaneously to their new positions. Separate calls to `Move To Cell` are issued for both actors. Additional checks ensure synchronization of animation timing and sound effects to preserve cinematic coherence.

En passant. For pawn captures en passant, the system identifies the captured pawn not by its destination square but by its original position in the previous rank. This ensures visual accuracy and prevents incorrect removal of actors.

Promotion. When the `Promotion?` flag is true, the pawn's mesh and metadata are replaced by those of the promoted piece. This substitution occurs immediately after the pawn reaches the destination cell, and the corresponding snapshot reflects the new piece identity.

4.1.6 Robustness and Limitations

Although the PGN parsing is designed to be general, it inherently assumes that the input notation follows standard algebraic format without comments or annotations. Cases involving ambiguous disambiguation (e.g., `Nbd2`) are resolved internally through the `Detect Piece` mechanism, which filters out illegal candidates. Nevertheless, the system does not attempt to validate moves globally or detect rule violations beyond local check conditions.

From an engineering standpoint, the choice to implement parsing in Blueprints (rather than in C++ or an external library) prioritizes visual clarity and project portability at the cost of some efficiency. This trade-off is justified by the limited scale of PGN files, typically under a few hundred moves, and by the pedagogical goals of this thesis, which emphasize transparency and accessibility of the logic within Unreal Engine's visual programming paradigm.

Through this architecture, the PGN parsing and move execution pipeline form a complete end-to-end transformation from symbolic notation to visual action. Each element, from regular expression parsing to snapshot validation, contributes to a unified system that ensures both cinematic fluidity and internal logical correctness.

This solid foundation supports the experimental work described in the following sections, where reinforcement learning is applied to optimize the aesthetic composition of the camera during match playback.

4.2 Cinematic Playback and Effects

The second subsystem of *Pawnimation* governs the playback of the chess match and the visual rendering of each move. This module transforms the abstract move data parsed from the PGN into coherent, cinematic 3D sequences, combining physical simulation, procedural animation, lighting, and sound design. All logic is implemented through Unreal Engine Blueprints, ensuring full integration with the engine's visual scripting environment and real-time rendering capabilities.

4.2.1 Playback Architecture

Once the PGN parsing phase has produced a complete list of `Move` structures, the system replays them sequentially within the level. For each move, the corresponding piece actor executes an animation while the environment, lighting, and camera remain dynamically responsive to the action.

The playback pipeline is orchestrated by the `Play Move` function, which receives both the current move and an animation duration as input. During normal playback (unlike the initial invisible run described in Section 4.1.3), animations are time-interpolated over the specified duration, enabling smooth cinematic motion and synchronization with sound and camera events.

The entire replay unfolds autonomously from the first to the last move, following the sequence encoded in the PGN file. During playback, the simulation proceeds continuously, with moves executed in order. However, the user can pause or resume the entire scene at any time, allowing temporary inspection or presentation control without disrupting the logical flow of the match.

4.2.2 Movement Animation Types

Two distinct motion profiles have been implemented for piece animations:

Linear glide. This is the default animation for most pieces, where the actor slides smoothly along the board surface from its starting to its destination cell. The movement is handled via a Blueprint timeline that interpolates position over time, producing a fluid, realistic gliding effect. This animation is accompanied by a short sound effect.

Arched trajectory. A more dynamic motion type is available for pieces that require elevated or compound movement paths. The piece follows a parabolic arc instead of a straight line, simulating a brief "jump." This motion is employed in two main cases:

- During castling, where one of the two pieces performs an arched motion while the other glides beneath it, resulting in a visually expressive coordinated animation.

- For knights, when both the straight-line and the L-shaped paths are obstructed. In this case, the system automatically switches to the arched trajectory to visually simulate the knight "jumping" over other pieces (Figure 4.4).

(a) Initial collision. **(b)** Partial fragmentation. **(c)** Final disintegration.

Figure 4.4. Progressive destruction of the Black Rook as the White Knight's capture unfolds through Chaos simulation.

This dual-animation system provides flexibility and helps maintain cinematic consistency, ensuring that even complex chess maneuvers are represented through visually engaging motion rather than purely functional translation.

4.2.3 Opening and Ending Cinematics

In addition to the per-move animations, *Pawnimation* introduces custom cinematic sequences at the beginning and end of each match, designed to enhance narrative immersion.

Opening cinematic. The initial sequence uses a Blueprint timeline to perform a panoramic orbit around the chessboard from a top-down perspective. The camera slowly descends and accelerates towards the board, culminating in a rapid "dive" into the scene. At the exact moment of this transition, a seamless switch occurs from the introductory camera to the reinforcement-learning-controlled camera, marking the beginning of the interactive playback phase. The transition is visually imperceptible, ensuring continuity between the pre-scripted and the AI-driven parts of the experience.

Ending cinematic. When the match concludes, if the result is not a draw, the camera automatically repositions to focus on the defeated King. A destruction effect is triggered in which the King's mesh implodes, symbolically representing checkmate and defeat. The sequence then fades into the game's ending screen, closing the narrative arc of the visualization.

4.2.4 Sound Design and Background Music

Sound plays a key role in reinforcing the cinematic dimension of the project. All sound assets are open-source and were integrated into Unreal Engine as individual `Sound Wave` components. Distinct audio effects were assigned to specific gameplay events:

- Sliding sound for standard ground movements.

- Takeoff and landing sounds for the arched jump animation.
- Explosion and shattering sounds for piece captures and destruction events.

Two musical themes were added as background tracks, both derived from an AI-generated composition. The first theme serves as the main menu soundtrack, evoking a calm yet anticipatory atmosphere. The second, used during gameplay, has a more dynamic and dramatic tone, supporting the tension of the unfolding match.

4.2.5 Visual Assets and Environment Lighting

3D Models and Asset Optimization All 3D models used for the board and pieces are open-source assets obtained from online repositories. These meshes were optimized for real-time rendering and physically accurate collisions, ensuring compatibility with the Chaos destruction system used during captures. Special attention was paid to maintaining a balance between polygon count and visual fidelity, so that the pieces could sustain dynamic destruction effects without compromising overall performance. The final result is a collection of lightweight yet detailed meshes suitable for cinematic visualization.

Environment and HDRI Background The environment background consists of an HDRI texture depicting a nebula in deep space, chosen to create a surreal, otherworldly atmosphere. The use of an HDRI not only provides photorealistic global illumination but also introduces soft environmental reflections on the glossy chessboard surface. The image's exposure was slightly increased to balance with the scene's internal illumination, avoiding excessive contrast between the bright chessboard and the dark background. This combination of cosmic imagery and physically based lighting reinforces the aesthetic vision of *Pawnimation*, which merges logic and fantasy into a cohesive cinematic world.

Lighting Configuration Lighting in the scene combines five main sources:

- The HDRI itself, which provides a diffuse global light contribution.
- Four diagonal directional lights, each oriented at a 45° angle from one corner of the board towards its center.

This configuration ensures uniform brightness across the scene while preserving soft shadows and subtle specular highlights on the reflective chessboard surface. Such an arrangement minimizes flat lighting artifacts while maintaining a sense of depth and volume in the scene. The overall effect is a bright, cinematic aesthetic that supports both readability and emotional engagement, particularly during capture and destruction sequences.

Main Menu Background The background image used in the main menu was derived directly from the in-game environment. A screenshot was captured during gameplay to preserve the original lighting and perspective of the scene (Figure 4.5). This screenshot was then provided as a reference to a generative AI model, which produced a cinematic reinterpretation of the same composition (Figure 4.6). The

generated image enhances visual depth and atmosphere while remaining faithful to the original environment, thereby ensuring a consistent and immersive visual identity between the game and its interface.

Figure 4.5. In-game screenshot captured during gameplay, used as the visual base for the main menu background.

Figure 4.6. AI-generated cinematic reinterpretation of the main menu background, based on the in-game environment.

Integrated Cinematic Experience Through this integrated animation and rendering pipeline, *Pawnimation* transforms a symbolic chess match into a vivid audiovisual performance. By combining procedural animation, physical destruction, dynamic lighting, and sound, the system achieves a film-like experience that bridges the gap between game replay and cinematic storytelling. Each visual element, from the optimized meshes to the spatial lighting composition, contributes to an immersive experience where strategic logic meets artistic expression.

4.3 Camera Control via Reinforcement Learning

Cinematic camera control is a critical aspect of interactive media, particularly in applications where visual storytelling and audience engagement are paramount. Traditional camera control systems in computer games and simulations typically rely on pre-scripted paths, spline-based interpolations, or manual operator input. While these methods can produce visually appealing results under controlled conditions, they are inherently limited in dynamic or unpredictable environments, such as the real-time replay of a chess match with full 3D piece animations.

The primary goal of this work is to develop a reinforcement learning (RL) framework capable of autonomously controlling a cinematic camera in a virtual chess environment. By leveraging RL, the system can learn to select camera positions and orientations that maximize visual quality and scene comprehension, without relying on manually authored trajectories. This approach enables the camera to dynamically adapt to diverse game states, capturing the essential actions of each move while maintaining a visually coherent and aesthetically pleasing composition.

In this context, the camera agent observes the current state of the chessboard, including the locations of the pieces involved in the move, their bounding volumes, and the midpoint between the starting and ending cells of the move. The agent executes continuous actions that control its movement along the forward and lateral directions relative to the camera's orientation, as well as its rotational orientation around the vertical axis. In addition, the agent outputs a discrete "Done" action, signaling the decision to halt motion once an optimal viewpoint has been achieved.

The design of the reward function is crucial in guiding the agent's behaviour. Positive rewards are granted for centering the midpoint of the move within the camera frustum, ensuring visibility of all relevant piece bounding boxes, and maintaining aesthetic alignment with the chessboard. Conversely, penalties are applied when parts of the pieces fall outside the camera frustum, when collisions or occlusions occur, or when the midpoint is poorly centered. The reward scheme is intentionally biased so that penalties outweigh rewards, discouraging premature termination and incentivizing the agent to continue adjusting until an optimal viewpoint is found (Figures 4.7 and 4.8).

Figure 4.7. Example of an intermediate camera position where the ending cell (EC), located behind the black pawn, is still partially occluded. The agent continues to adjust its position, keeping the "Done" flag set to *false* while searching for a better viewpoint.

Figure 4.8. Final camera position where the ending cell (EC), previously hidden behind the black pawn, becomes fully visible. The agent recognizes the satisfactory viewpoint, sets the "Done" flag to *true*, and the move animation (in this case, a bishop move) begins.

4.3.1 Problem Formulation

The task of cinematic camera control for chess move visualization can be formulated as a sequential decision-making problem, which naturally fits within the framework of reinforcement learning. In this setting, the camera agent interacts with a virtual environment representing the chessboard, the pieces, and the current move.

At each discrete time step t, the agent observes a state s_t, executes an action a_t, and receives a reward r_t that quantifies the quality of the resulting camera configuration.

State Space

The state s_t encodes all information necessary for the agent to make informed decisions about camera positioning and orientation. In this system, the state includes:

- The world-space positions of the starting cell (SC) and ending cell (EC) of the current chess move.
- The midpoint MP between SC and EC, elevated to the camera plane height, which serves as the primary visual target.
- The bounding box points of the pieces involved in the move, providing information about spatial extent: the points of the piece in the starting cell (SBB, Start Bounding Box) and the piece in the ending cell (EBB, End Bounding Box).
- Ray observations from the camera towards SC and EC, encoding potential occlusions and line-of-sight information.
- The current camera position and rotation, represented as a vector (x, y, z, θ), allowing the agent to reason about relative motion and angular adjustments.

To gather all this information at runtime, a dedicated Blueprint function named `Gather Observations` was implemented. This function aggregates spatial, geometric, and perceptual data from multiple in-game sources into a single structured representation. Due to the large number of nodes required to collect and process each observation component, the resulting Blueprint graph is extensive and visually dense, as shown in Figure 4.9. The figure highlights the structural complexity of the observation pipeline, emphasizing the detailed integration effort required to feed the reinforcement learning agent with accurate environmental data.

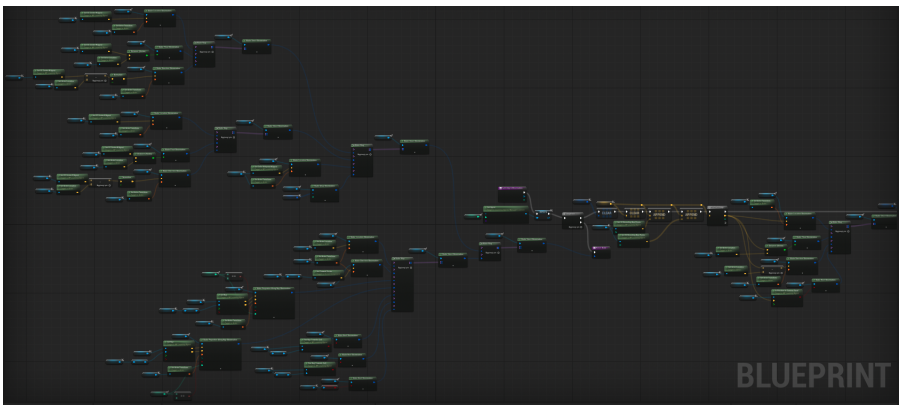

Figure 4.9. Blueprint implementation of the `Gather Observations` function. The graph is highly dezoomed to illustrate its structural complexity rather than individual node logic.

Formally, the state can be written as:

$s_t = \{\text{SC position}, \text{EC position}, MP, \text{SBB points}, \text{EBB points}, \text{ray observations}, \text{camera pose}\}$.

Action Space

The agent's action a_t consists of continuous and discrete components controlling camera movement and decision-making:

- **Continuous Actions:**
  - Forward speed v_f, controlling motion along the camera's forward vector.
  - Side speed v_s, controlling lateral movement along the camera's right vector.
  - Rotation $\Delta\theta$, controlling yaw around the vertical axis.
- **Discrete Action:**
  - *Done* $\in \{0, 1\}$, a boolean signal indicating whether the camera should stop moving.

These actions are applied to the camera in world space, allowing flexible adjustments in position and orientation. The *Done* action provides a mechanism for the agent to determine when a visually satisfactory viewpoint has been achieved, solving the oscillation problem observed in naive continuous control.

Reward Function

The reward r_t is designed to guide the agent towards optimal cinematic viewpoints. It is composed of multiple components that reflect the visual quality and completeness of the framing:

- **Midpoint Proximity Reward:** Positive reward inversely proportional to the distance between the camera and the midpoint MP, encouraging close-up views of the action.
- **Midpoint Centering Reward:** Positive reward proportional to the alignment of MP with the center of the camera frustum.
- **Bounding Box Visibility Penalty:** High penalty when any corner of SBB or EBB is outside the camera frustum, ensuring all relevant pieces remain visible.
- **Raycasting Occlusion Penalty:** Additional penalty if raycasts detect obstacles obstructing the view of SC or EC.
- **Done Reward/Penalty:** If the agent executes the *Done* action while all other penalties have been eliminated, a large positive reward is granted. Otherwise, a significant penalty discourages premature stopping.

The cumulative reward R_t encourages the agent to move dynamically, maintain clear visibility of the move, and stop only once an optimal framing has been achieved:

$$R_t = r_{\text{proximity}} + r_{\text{MP}} - r_{\text{occlusion}} - r_{\text{frustum}} + r_{\text{done\_bonus/penalty}}.$$

4.3.2 Agent Design

The reinforcement learning agent controlling the cinematic camera in *Pawnimation* was implemented entirely within Unreal Engine 5 using Blueprints. The decision to employ a Blueprint-based implementation stemmed from the need for tight integration with the 3D environment and real-time feedback from visual and spatial components. This allowed the agent to directly manipulate the camera actor and receive perceptual feedback through the engine's native systems, ensuring seamless synchronization between learning, movement, and rendering.

The agent operates as a specialized *Pawn* within Unreal Engine, representing the camera that moves and rotates on the board plane. Its purpose is to find, for each chess move, a cinematic position that maximizes visual clarity and aesthetic composition. The core of its behaviour lies in the function `PerformAgentAction`, which executes the actions predicted by the neural network. These actions correspond to the continuous control of the camera's translation and rotation, plus a discrete boolean signal that determines when the agent decides to stop moving and fix the final framing.

Action Execution Pseudocode

The agent's action space is composed of four elements:

1. **Forward Speed:** controls the camera's movement along its local forward axis.
2. **Side Speed:** controls lateral movement (right-left) relative to the current facing direction.
3. **Z Rotation:** determines the rotation of the camera around its vertical axis, allowing panning across the scene.
4. **Done Flag:** a boolean action that signals the decision to stop moving and lock the camera position.

The first three elements are continuous-valued outputs from the neural network, while the fourth is a discrete signal. The overall action vector, therefore, combines both continuous and discrete components. In each step, the environment feeds this action vector to the `PerformAgentAction` function, which applies it to the in-game camera pawn.

The pseudocode governing this execution is shown below:

```
// Function PerformAgentAction
// Input: Agent Id

CALL GetAgent with Agent Id
SET ActActor TO the returned agent

CALL GetStructAction with Tag: StructAction
SET OutElements TO the returned map of elements

SEQUENCE
```

```
    CALL GetFloatAction with Tag: Forward Speed
    SET ActActor's Forward Speed TO returned value

    CALL GetFloatAction with Tag: Side Speed
    SET ActActor's Side Speed TO returned value

    CALL GetFloatAction with Tag: Z Rotation
    SET ActActor's Z Rotation TO returned value

    CALL GetBoolAction with Tag: BoolAction
    SET DoneValue TO returned boolean value

    IF (NOT ActActor's Done) AND DoneValue THEN
        SET ActActor's Done to DoneValue
        CALL Camera Done on ActActor
    ENDIF
END SEQUENCE
```

Movement Control and Stopping Criterion

The three continuous actions control the camera's position in real time. Forward and side speeds are applied as additive movement inputs within the pawn, while the Z rotation updates the actor's yaw, producing smooth panning motions. However, the true innovation lies in the **Done** flag. This boolean, produced directly by the neural network, determines when the camera should stop exploring and fix its final position.

When the `Done` flag is set to true, the camera ceases all movement and rotation, effectively "locking" in the current viewpoint. This mechanism is crucial: without it, the agent would continue to oscillate indefinitely, trying to marginally improve the reward by small lateral adjustments. Early versions of the system exhibited precisely this behaviour - a constant trembling motion even after reaching an optimal vantage point. Introducing the `Done` flag allowed the network to explicitly learn when to stop, transforming the control problem from pure continuous navigation into a hybrid continuous–discrete decision-making task.

Internal State and Inhibition Logic

The `Done` variable is also stored as an internal state within the agent blueprint. Once set to `true`, this variable inhibits all future movement updates until the next environment reset. This ensures temporal consistency: the camera remains completely still until the chess move finishes and the next one begins. The Blueprint-level inhibition mechanism is simple yet effective, guaranteeing that the agent's output translates into stable cinematic shots without jitter or drift.

Motivation and behaviour Implications

This hybrid structure of actions enables a nuanced control strategy. The continuous components let the camera explore the space fluidly, while the discrete

component introduces a notion of intentional finality - the agent can express not only *how* to move but also *when* to stop. This distinction proved essential for achieving cinematographic stability.

The `Done` signal is also directly tied to the reward design (discussed in Section 4.3.4). Because the agent only receives a significant positive reward when stopping in a visually valid position, the model learns to associate the decision to halt with aesthetic correctness. Conversely, stopping prematurely or in a penalized position (e.g., when pieces are out of frame or occluded) leads to a sharp negative cumulative reward. This dynamic naturally encourages the agent to move just enough to eliminate penalties before deciding to stop.

In summary, the agent design combines intuitive, physically meaningful control parameters with a reinforcement signal that embeds cinematic reasoning. It represents a balance between freedom of motion and stability of framing - a critical requirement for achieving the film-like aesthetic that distinguishes *Pawnimation* from typical game camera controllers.

4.3.3 Environment Design

The training environment represents the physical and logical framework in which the reinforcement learning agent operates. In this project, the environment is implemented directly within Unreal Engine 5, leveraging both its visual rendering capabilities and its physics-based interactions. The goal of the environment is to provide the agent with a realistic yet controlled simulation space where camera movements, chessboard geometry, and piece positions interact coherently with the reward structure.

Environment Initialization

At the beginning of each training epoch two distinct reset procedures take place and they serve different purposes. The first procedure, triggered by the Level Blueprint event `RandomizeEnvironment`, is responsible for resetting the environment itself: it randomly selects and loads the next training move, updates the board snapshot and any move-specific state, and prepares the scene for the new episode. This `RandomizeEnvironment` event therefore controls which chess move the agent must frame during the upcoming episode.

The second procedure, implemented as the function `Reset To Random Point In Chessboard`, operates at the agent level: it reinitializes the spatial configuration of the camera agents inside the environment. For each instance of `LearningAgent` the function samples a candidate (x, y) position and a yaw rotation within the allowed bounds (e.g., ± 1500 Unreal units around the chessboard center and $[0, 360)$ degrees), sets the actor transform, and repeats the sampling until a non-overlapping placement is found. During this loop simple proximity checks prevent agents from spawning too close to one another, and linear/angular velocities are reset to zero to avoid residual motion artifacts.

Splitting the initialization responsibilities in this way - `RandomizeEnvironment` for the logical episode content (which move to present) and `Reset To Random Point In Chessboard` for agent spatialization - provides clear separation of concerns and

improves reproducibility.

Progressive Move Randomization

A distinctive feature of the environment is its probabilistic selection of the move type at the start of each episode. During training, the agent must learn to frame both short and long chess moves. Empirically, short moves (1–3 cells) are far more frequent and easier to capture, whereas long moves require the camera to move further away to achieve a balanced composition. To address this imbalance, a probability-based mechanism is introduced: at the beginning of each training episode, with probability p_{long} the environment selects a move from the *Long Moves* array, which contains all moves longer than four cells, as well as diagonal moves of length four.

The parameter p_{long} increases progressively throughout the training process, starting from 0.1 and reaching 0.5 over the course of the ten training games. This approach enables the agent to first master simpler, more common situations before being exposed to rarer and more visually demanding ones. It mirrors the principles of curriculum learning, where the difficulty of the environment gradually scales with the agent's capabilities.

Agent–Environment Interaction

Once the initial conditions are set, the environment continuously interacts with the agent during each simulation tick. The agent issues three continuous action commands - `Forward Speed`, `Side Speed`, and `Z Rotation` - which directly control the motion of its associated camera actor. These values are applied to the actor via Unreal Engine's native `AddMovementInput` and `AddActorWorldLocation` nodes.

An additional discrete action, the `Done` flag, allows the agent to signal that it has found a satisfactory camera position. When this flag is set to true, all movement inputs are disabled, effectively freezing the camera in place. This mechanism is essential for achieving convergence and preventing oscillatory behaviour once an optimal or near-optimal viewpoint is reached.

Visual Metrics and Scene Geometry

The primary spatial entities involved in the reward calculation are:

- The **starting cell (SC)** and **ending cell (EC)** of the chess move, represented by their 3D world coordinates and bounding boxes (SBB and EBB);
- The **midpoint (MP)** between SC and EC, defined as

$$MP = \frac{SC + EC}{2},$$

  which serves as the optimal target for the camera to center on.

The camera's goal is to position itself such that both bounding boxes (SBB and EBB) are visible within the frustum, while also keeping the midpoint (MP) near the center of the screen. To achieve this, the environment computes for each frame:

1. The normalized screen-space distance between the projected midpoint and the center of the viewport;
2. The percentage of bounding box corners lying within the frustum;
3. The visibility (via raycasting) of the bounding box centers.

These metrics are not only used for reward computation but also for debugging and performance visualization during training, allowing developers to track the camera's framing quality over time.

Scalability and Synchronization

The environment is designed to operate in tight synchronization with the Proximal Policy Optimization (PPO) trainer and the Unreal Engine tick system. Each tick represents a discrete time step in the Markov Decision Process (MDP). The environment collects the current state, applies the agent's action, and computes the resulting reward, which is then passed to the trainer through the socket-based communication layer. This tight coupling between Unreal's real-time loop and the PPO training cycle ensures deterministic updates and reproducible learning behaviour.

Furthermore, multiple agent instances can be spawned within the same environment. Each agent is initialized with a different starting transform, allowing parallelized experience collection that accelerates learning.

4.3.4 Reward Function Design

A crucial aspect of reinforcement learning lies in the design of the reward function, which encodes the objectives the agent must pursue and, consequently, determines the resulting behaviour. In the context of *Pawnimation*, the goal of the camera-controlling agent is not simply to track the pieces' movements functionally, but rather to produce cinematographically pleasing shots that highlight the action dynamically and aesthetically. Therefore, the reward design aims to align the agent's learning process with both geometric correctness and artistic appeal.

Cinematic Motivation

In traditional camera control for games or simulations, rewards are often defined by visibility and framing constraints: the subject should remain visible and centered within the camera's field of view. However, in this project, the agent's reward is explicitly formulated to encourage a cinematic sensibility. The reward structure integrates notions such as composition balance, framing of the action midpoint, and avoidance of occlusions, thereby enabling the camera to behave as a virtual cinematographer rather than a tracking sensor.

The camera is positioned at human scale on the chessboard, with pieces modeled as life-sized sculptures. Each move in the replay therefore resembles a physical event in a large environment. The reward encourages the agent to choose viewpoints that make the action readable yet visually interesting - for example, capturing the movement of a queen diagonally across the board from a low angle, or framing the

destruction of a captured piece from a dramatic side perspective. This transforms the purely spatial task into a visually driven optimization problem.

Mathematical Structure of the Reward

The camera reward is computed as a weighted sum of several distinct terms that evaluate both geometric correctness and cinematic qualities. Rather than enforcing a single scalar objective, the function aggregates complementary signals - hard constraints, softer penalties, and positive incentives - that together shape the agent's behaviour. The total instantaneous reward at time t is therefore given by the sum of the following contributions:

$$R_t = R_{\text{boundary}} + R_{\text{visibility}} + R_{\text{bbox\_coverage}} + R_{\text{mp\_closeness}} + R_{\text{mp\_centrality}} + R_{\text{done}}$$

Below we describe each term and the design rationale.

Boundary penalty R_{boundary}. A hard spatial constraint discourages the camera from leaving the valid operational region. If the camera's displacement from the chessboard center exceeds a predefined threshold, a large negative penalty is applied. This acts as a safety term that keeps exploration inside the playable / cinematic workspace and prevents pathological solutions where the camera moves arbitrarily far away. The effective contribution is of the form:

$$R_{\text{boundary}} = \begin{cases} -100.0, & \text{if } \|\mathbf{p}_{\text{cam}} - \mathbf{p}_{\text{cc}}\| > 1900 \\ 0, & \text{otherwise} \end{cases}$$

where $\mathbf{p}_{\text{cam}}$ is the camera position, $\mathbf{p}_{\text{cc}}$ is the chessboard center position and the threshold is 1900 units (Unreal units). The large magnitude ensures this term dominates when violated.

Visibility penalties $R_{\text{visibility}}$. Visibility of the move endpoints is enforced through two primary boolean penalties and an additional global penalty when either endpoint is not visible:

$$R_{\text{visibility}} = -\mathbb{1}_{\text{SC not visible}} \cdot 1.0 \; - \; \mathbb{1}_{\text{EC not visible}} \cdot 1.0 \; - \; \mathbb{1}_{\text{SC or EC not visible}} \cdot 3.0,$$

where $\mathbb{1}$ denotes the indicator function evaluating to 1 when the specified condition is true. Each missing visibility (Start Cell or End Cell) incurs an individual penalty of magnitude 1.0, and an additional penalty of 3.0 is applied whenever at least one of the two endpoints is not visible. This design strongly enforces the narrative requirement that both key elements of the move remain within the viewer's sight.

Bounding-box coverage penalties $R_{\text{bbox\_coverage}}$. To enforce that the visual extent of the moving pieces is within the frame, two related penalties are applied:

1. A per-corner penalty proportional to the total number of bounding-box corner points that fall outside the camera view. If N_{out} is the number of out-of-view corner points across both SBB and EBB, then

$$R_{\text{corners}} = -0.125 \times N_{\text{out}}.$$

   This provides a graded penalty: each missing corner reduces the reward by a small amount, encouraging the agent to progressively correct partial framings.

2. A strong penalty if neither SBB nor EBB is fully contained in the frustum. In other words, if both bounding boxes are not fully in view, an additional discrete penalty is applied:

$$R_{\text{full\_view\_fail}} = \begin{cases} -3.0, & \text{if both SBB and EBB are not fully in view} \\ 0, & \text{otherwise.} \end{cases}$$

The combination of these terms produces both a coarse gating effect (the -3.0 penalty) and a fine-grained corrective pressure (the -0.125 per corner).

Midpoint closeness reward $R_{\text{mp\_closeness}}$. Cinematic composition is encouraged by a continuous reward that increases as the camera approaches the geometric midpoint between SC and EC (projected at camera height). This term is implemented as a location-similarity measure with a distance scale parameter and a bounded amplitude. Practically, the term can be expressed as a monotonically decreasing function of the camera–midpoint distance $d = \|\mathbf{p}_{\text{cam}} - \mathbf{p}_{\text{MP}}\|$, normalized by a characteristic length L and scaled by a maximum amplitude A:

$$R_{\text{midpoint\_closeness}} = f_{\text{sim}}(d; L) \quad \text{with} \quad f_{\text{sim}}(0; L) \approx 1.5,$$

where in the implemented configuration the distance scale L is on the order of 1000 units and the peak contribution is approximately $+1.5$. The precise kernel is selected to provide a smooth, bounded incentive for closeness without overwhelming the hard visibility constraints.

Midpoint centrality reward $R_{\text{mp\_centrality}}$. A small positive term further encourage centering relevant elements in the frame:

$$R_{\text{midpoint\_centrality}} = +0.5 \cdot \mathbb{1}_{\text{MP centered}}$$

It contributes modestly ($+0.5$) if the midpoint is centered, nudging the agent toward globally coherent compositions.

Done bonus R_{done}. To further encourage the agent to identify aesthetically satisfying viewpoints, a special stopping reward multiplier is introduced. When the agent decides to terminate its movement - that is, to "lock" the camera position before the next move occurs - all rewards are multiplied by a factor of 11. This creates an incentive to stop only in frames that are both geometrically valid and

cinematographically meaningful. Empirically, this mechanism prevents oscillatory behaviour and leads to stable, well-composed final shots.

Formally:

$$R'_t = \begin{cases} 11 \times R_t, & \text{if } a_t = \text{stop} \\ R_t, & \text{otherwise} \end{cases}$$

This modification transforms the standard episodic RL problem into a hierarchical one, where the agent alternates between exploration (moving to improve framing) and exploitation (committing to a final composition). The stopping decision acts as a self-evaluation of the cinematic adequacy of the shot.

Net effect and design rationale. The chosen magnitudes create a strong asymmetry between penalties and rewards: hard failures (obstructed line of sight, lack of complete bounding-box coverage) produce large negative values, while positive incentives are comparatively small (except the Done bonus). This asymmetry was intentionally designed so that the agent will incur a net negative cumulative reward unless all major constraints are satisfied. Consequently:

- The agent cannot obtain the Done bonus in a profitable manner unless it first eliminates visibility and framing penalties.
- The per-corner penalty provides a smooth gradient to guide incremental improvement toward full coverage.
- The midpoint proximity reward and the small in-view incentives steer the agent toward aesthetically preferable viewpoints once hard constraints are met.

In practice, this composition produces the desired emergent behaviour: the agent explores until visibility and coverage constraints are satisfied, only then activating the positive Done bonus and thereby stabilizing on a final, non-oscillatory camera pose. The heavy boundary penalty additionally ensures that the agent remains confined to a sensible operational region of the scene.

Implementation Details

In Unreal Engine 5, these rewards are computed at runtime using Blueprint nodes combined with raycasting, frustum inclusion checks, and geometric calculations. The midpoint and bounding boxes are extracted from the move data parsed from the PGN file, while visibility and occlusion tests are executed via Unreal's physics and rendering subsystems.

To maintain performance during training, the entire evaluation is implemented using lightweight vector math and conditional logic rather than visual scripting for loops. This ensures that the agent can perform thousands of iterations per move sequence without perceptible delay.

Qualitative Outcomes

Through this reward design, the agent learns to select positions that mimic human cinematographic intuition. It avoids occluded or misaligned angles, prefers

slightly offset shots that convey depth, and often centers the midpoint of the action in the frame while ensuring both pieces are visible. When combined with the destruction effects and spatial lighting, the resulting sequences convey a sense of tension and spectacle, transforming a chess replay into a narrative visual experience.

This reward-driven aesthetic behaviour represents the core originality of the project: instead of explicitly coding camera trajectories or using handcrafted rules, the system learns from feedback shaped by geometric and cinematic criteria. The reinforcement learning paradigm thus becomes a bridge between computational optimization and artistic composition.

4.3.5 Learning Algorithm

The reinforcement learning component of the system is based on the *Proximal Policy Optimization* (PPO) algorithm [14], a widely used policy-gradient method known for its stability and efficiency in continuous control tasks. PPO strikes a balance between exploration and exploitation by constraining policy updates through a clipped objective, preventing drastic deviations from previously successful policies.

Inference runs deterministically with a noise scale of zero, ensuring stable camera behaviour during cinematic playback. Synchronization between the Unreal Engine environment and the PPO trainer is maintained through socket-based communication, with frame updates handled on a per-tick basis to ensure real-time responsiveness and consistent temporal coordination.

Chapter 5

Experiments and Discussion

The reward function designed for the cinematic camera agent is intentionally complex, combining multiple terms that jointly evaluate geometric accuracy and aesthetic quality. However, to assess the true contribution of each component, an ablation experiment was conducted. The goal of this experiment was to determine how the presence or absence of specific reward terms influences the agent's ability to produce satisfactory cinematic shots.

In particular, several "incomplete" agents were trained, each with a modified version of the original reward function where one or more criteria were deliberately removed. For example, some agents ignored the closeness reward, others did not account for midpoint centering, and others omitted visibility constraints. This approach makes it possible to isolate the behavioural effect of each reward component and to identify which terms are essential for coherent camera control.

5.1 Experimental Design

To systematically evaluate the contribution of individual reward components, four "incomplete" agents were created by selectively removing specific terms from the full reward function. Each agent was trained under identical environmental conditions to ensure comparability. The modifications were as follows:

- **No Midpoint Closeness Reward**: The reward term encouraging the camera to remain near the move midpoint was removed, leaving all other terms active.
- **No Midpoint Centrality Reward**: The reward for keeping the move midpoint centered in the frame was omitted.
- **No Visibility Penalty**: The penalties associated with the starting cell and the ending cell being occluded were disabled.

In each case, the remaining active reward terms were rescaled to preserve the fundamental property of the reward function: unless all penalties were resolved, the cumulative reward remained negative, preventing the agent from prematurely activating the "Done" flag. This normalization ensures that the observed behavioural differences are attributable solely to the removal of specific reward criteria, rather than to changes in overall reward magnitude.

All agents were trained for a fixed number of episodes, using identical randomization of initial camera positions and moves. Metrics were collected throughout training, providing quantitative insight into the influence of each reward component on the agent's ability to generate cinematic camera behaviour.

5.2 Results and Analysis

This section presents the outcomes of the ablation experiment conducted to evaluate the contribution of each individual reward term to the agent's overall behaviour. The experiment was performed on a set of twenty test matches, each comprising multiple moves. For each match, the average performance of thirty-two independent agents of the same type was recorded, ensuring statistical robustness of the obtained measures.

Four different reward configurations were tested: the *All Rewards* agent (full configuration), and three *truncated* agents.

Each agent was evaluated along four metrics:

1. **Average Distance from the Midpoint (lower is better)** – measures how close the camera remains to the midpoint between the starting and ending cells.
2. **Average Midpoint Distance from Screen Center (lower is better)** – measures how close the midpoint projection lies to the screen center.
3. **Average Start Cell Visibility (higher is better)** – binary measure (1 if visible, 0 otherwise) averaged over all moves.
4. **Average End Cell Visibility (higher is better)** – analogous to the above for the destination cell.

Test Configuration. For each match, four curves were plotted (one per agent type) for each metric, with the x-axis representing the move number and the y-axis showing the averaged value among the 32 instances. The average of these values across all moves yields the global performance score reported below.

Quantitative Results on Test Match 1

Table 5.1 summarizes the global averages for the four agents on the first test match. For readability, the agent types have been abbreviated as follows: **AllRew** (All Rewards), **No CloRew** (No Midpoint Closeness Reward), **No CenRew** (No Midpoint Centrality Reward), and **No VisPen** (No Visibility Penalty). The observed values are consistent with those recorded across the remaining nineteen test matches, confirming the reliability of the trends described herein.

Table 5.1. Average performance metrics for the four agents on Test Match 1.

Agent	Camera-MP Dist.	MP–Center Dist.	SC Vis.	EC Vis.
AllRew	749	0.11	0.94	0.97
No CloRew	1063	0.06	0.90	0.95
No CenRew	679	0.18	0.92	0.98
No VisPen	665	0.06	0.76	0.75

To complement the numerical results reported in Table 5.1, Figures 5.1–5.4 depict the performance of the four agents on each move of Test Match 1. Each curve represents the mean value across 32 independent instances of the same agent type. This visualization allows for a more detailed examination of the agents' behaviour over time and provides insights into how each reward term influences the emergent camera control strategy.

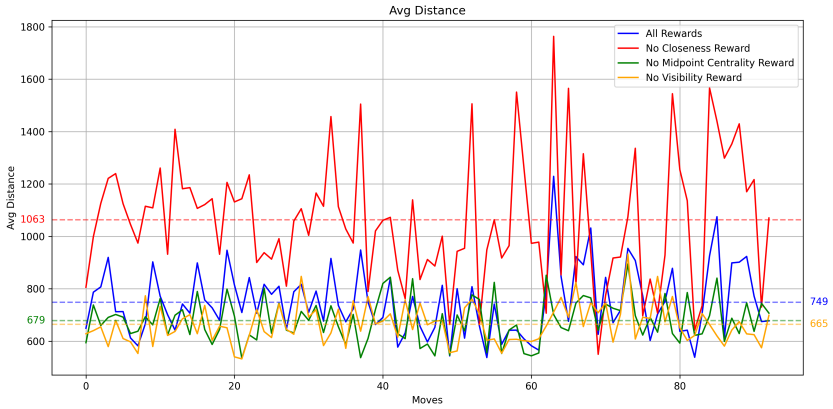

Figure 5.1. Average Distance from the Midpoint per move for Test Match 1. Each curve represents the mean value across 32 instances of each agent type. Higher values indicate worse performance.

Figure 5.1 shows that the **No CloRew** agent exhibits the highest average distance from the move midpoint, confirming that the midpoint closeness reward is essential for maintaining proximity. Interestingly, the **No VisPen** and **No CenRew** agents achieve slightly lower distances than the full **AllRew** agent, illustrating that removing one reward allows the agent to better optimize the remaining objectives due to fewer constraints.

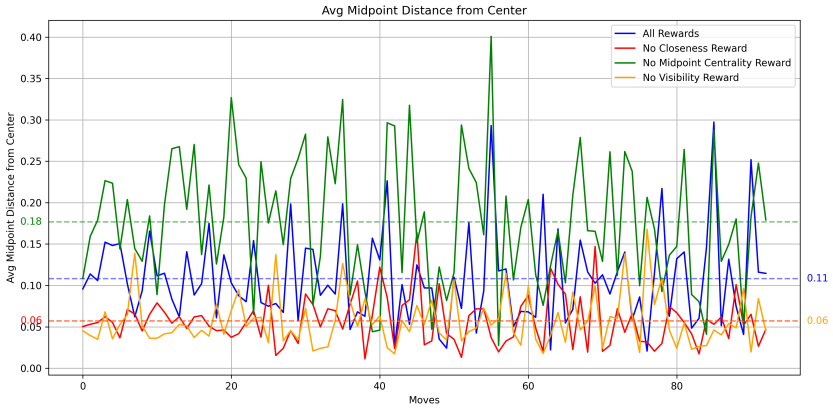

Figure 5.2. Average Midpoint Distance from Center per move for Test Match 1. Higher values indicate poorer centrality.

Figure 5.2 confirms that the **No CenRew** agent performs worst in keeping the midpoint centered on screen, as expected. The **No CloRew** and **No VisPen** agents perform better than the full agent, again reflecting the effect of fewer constraints, allowing them to prioritize midpoint centrality more effectively.

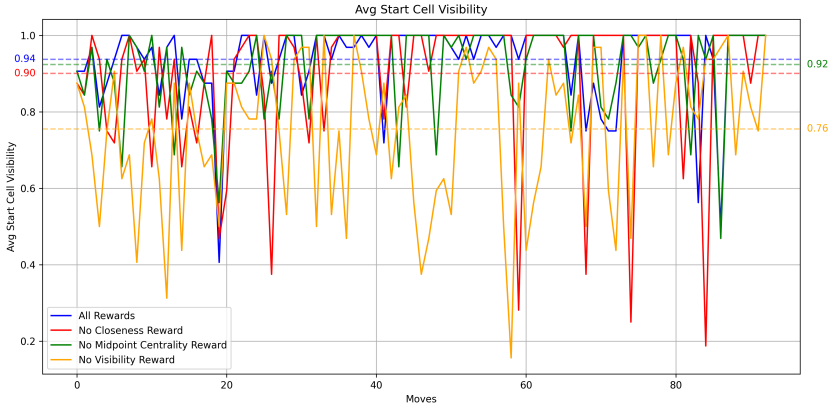

Figure 5.3. Average Start Cell Visibility per move for Test Match 1. Higher values indicate better visibility.

Figure 5.3 illustrates that removing the visibility penalty (**No VisPen**) drastically reduces start cell visibility, validating the importance of raycasting-based penalties. The other truncated agents maintain visibility close to the full agent, demonstrating that their modifications do not significantly impair this metric.

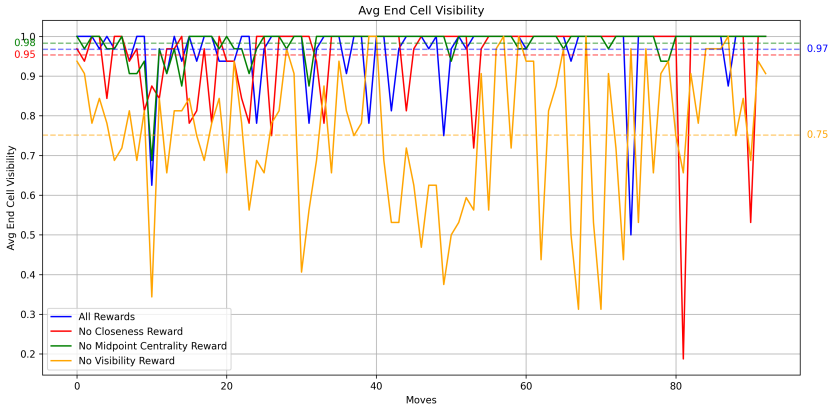

Figure 5.4. Average End Cell Visibility per move for Test Match 1. Higher values indicate better visibility.

Figure 5.4 shows a pattern similar to start cell visibility. The **No VisPen** agent performs worst, while the **No CenRew** agent slightly outperforms the full agent, highlighting that relaxing one constraint allows the agent to more effectively optimize other objectives.

Discussion. Across all metrics, the ablation results consistently show that an agent lacking a specific reward performs worst on that particular metric. Conversely, truncated agents can outperform the full agent on other metrics, as removing one reward reduces the number of competing objectives and allows the agent to better optimize the remaining terms. This phenomenon demonstrates the interdependence of reward terms and how the careful balance of constraints influences emergent behaviour. These trends were observed across all twenty test matches, confirming their generality and robustness.

Chapter 6

Conclusions and Future Work

6.1 Conclusions

This thesis presented *Pawnimation*, a cinematic chess visualization system in Unreal Engine 5, where camera control is learned via Reinforcement Learning. The proposed agent successfully learned to autonomously choose camera positions that are both visually pleasing and technically correct, balancing multiple reward components such as midpoint proximity, bounding box visibility, and framing within the camera frustum.

Ablation studies demonstrated the importance of each reward term: removing a specific component consistently degraded the agent's performance on the corresponding metric, while occasionally improving performance on other metrics due to reduced constraints. These findings validate the design of the reward function and confirm the emergent behaviour of the agent in dynamically selecting aesthetically pleasing camera angles.

Overall, the system achieves the goal of generating cinematic replays of chess games without manual scripting, and the reinforcement learning framework provides a flexible basis for further extensions.

6.2 Future Work

Several avenues exist to enhance the agent's capabilities and the overall cinematic experience:

6.2.1 Reinforcement Learning Improvements

- Extend raycasting penalties to all points of the bounding boxes, instead of only at the center of the cells, analogous to the existing frustum-based checks.
- Apply visibility penalties (both raycasting and frustum) to all cells along the piece's trajectory, rather than only the starting and ending cells.
- Allow the agent to rotate along the Y-axis and move vertically (up and down), increasing the range of potential camera perspectives.

6.2.2 User Experience and Stylistic Enhancements

- Introduce a pause menu to control playback of the cinematic replay.
- Provide options to select different background audio tracks or visual environments according to the desired mood or context.
- Enable more advanced playback control, such as skipping directly to a specific move or returning to the previous one.
- Enable direct input of a chess game link, automating PGN extraction and facilitating immediate visualization.

These improvements would not only expand the agent's operational capabilities but also enhance the user experience, allowing for richer, more immersive, and easily customizable cinematic presentations of chess games.

Bibliography

[1] Amerson, D., Kime, S., and Young, R. M. Real-time cinematic camera control for interactive narratives. In Proceedings of the 2005 ACM SIGCHI International Conference on Advances in computer entertainment technology, pp. 369–369 (2005).

[2] Arulkumaran, K., Deisenroth, M. P., Brundage, M., and Bharath, A. A. Deep reinforcement learning: A brief survey. IEEE signal processing magazine, **34** (2017), 26.

[3] Azzarelli, A., Anantrasirichai, N., and Bull, D. R. Intelligent cinematography: a review of ai research for cinematographic production. Artificial Intelligence Review, **58** (2025), 108.

[4] Dhahir, M. Q., Al-Hakeem, M. A. M., and Alshadoodee, H. Automatic camera control and artificial intelligence in the future of cinematography. International Journal, **7** (2022).

[5] Fang, H., Liu, H., Wen, J., Yang, Z., Li, J., and Han, Q. Automatic visual enhancement of ptz camera based on reinforcement learning. Neurocomputing, **626** (2025), 129531.

[6] Galvane, Q., Ronfard, R., Christie, M., and Szilas, N. Narrative-driven camera control for cinematic replay of computer games. In Proceedings of the 7th International Conference on Motion in Games, pp. 109–117 (2014).

[7] Harmon, M. E. and Harmon, S. S. Reinforcement learning: A tutorial. (1997).

[8] He, H., Xu, Y., Guo, Y., Wetzstein, G., Dai, B., Li, H., and Yang, C. Cameractrl: Enabling camera control for video diffusion models. In The Thirteenth International Conference on Learning Representations (2025).

[9] Kaelbling, L. P., Littman, M. L., and Moore, A. W. Reinforcement learning: A survey. Journal of artificial intelligence research, **4** (1996), 237.

[10] Li, Y. Deep reinforcement learning: An overview. arXiv preprint arXiv:1701.07274, (2017).

[11] Mousavi, S. S., Schukat, M., and Howley, E. Deep reinforcement learning: an overview. In Proceedings of SAI intelligent systems conference, pp. 426–440. Springer (2016).

[12] Passalis, N. and Tefas, A. Deep reinforcement learning for controlling frontal person close-up shooting. Neurocomputing, **335** (2019), 37.

[13] Pickel, B. and Rabern, B. Scorekeeping in a chess game. Semantics and Pragmatics, **15** (2022), 12.

[14] Schulman, J., Wolski, F., Dhariwal, P., Radford, A., and Klimov, O. Proximal policy optimization algorithms. arXiv preprint arXiv:1707.06347, (2017).

[15] Shakya, A. K., Pillai, G., and Chakrabarty, S. Reinforcement learning algorithms: A brief survey. Expert Systems with Applications, **231** (2023), 120495.

[16] Stallings, L. A brief history of algebraic notation. School Science and Mathematics, **100** (2000), 230.

[17] Wisniewski, M., Rana, Z. A., and Petrunin, I. Reinforcement learning for pan-tilt-zoom camera control, with focus on drone tracking. In AIAA SCITECH 2023 Forum, p. 0194 (2023).

[18] Wu, X., Wang, H., and Katsaggelos, A. K. Automatic camera trajectory control with enhanced immersion for virtual cinematography. arXiv preprint arXiv:2303.17041, (2023).

[19] Yang, Z., Liu, H., Fang, H., Li, J., and Jiang, Y. Multi-agent hierarchical reinforcement learning for ptz camera control and visual enhancement. Electronics, **14** (2025), 3825.