Universiteit Leiden

Master Media Technology

Mirror Mode: Your Biggest Enemy is Yourself

Name:	Yanna Elizabeth Smid
Student ID:	s2634163
Date:	27/08/2025
1st supervisor:	Dr. P.W.H. van der Putten, Leiden University
2nd supervisor:	Prof.dr. A. Plaat, Leiden University

Master's Thesis in Media Technology

Leiden Institute of Advanced Computer Science
Leiden University
Einsteinweg 55
2333 CC Leiden
The Netherlands

Mirror Mode: Your Biggest Enemy Is Yourself

Yanna E. Smid
y.e.smid@umail.leidenuniv.nl

Master's Thesis in Media Technology
Leiden Institute of Advanced Computer Science (LIACS)
Leiden University

Primary supervisor:
Dr. P.W.H. van der Putten, Leiden University

Secondary supervisor:
Prof.dr. A. Plaat, Leiden University

Abstract. Turn-based strategy games face a decline in popularity, possibly caused by repetitive gameplay and predictable enemy strategies. Therefore, this study introduces Mirror Mode, a new game mode where the enemy AI mimics the personal strategy of a player. A simplified version of the Nintendo strategy video game Fire Emblem Heroes has been built for this in Unity, with a Standard Mode and a Mirror Mode. Preliminary experiments have been carried out to design a suitable model for the task to imitate player demonstrations. A configuration of Reinforcement Learning and Imitation Learning techniques is used for this. The second set of experiments evaluates the constructed model during player tests, where models are trained specifically on the demonstrations provided by participants. Results find potential signs of copied defensive strategies, but still lack in imitating offensive strategies. Participants generally show higher satisfaction scores for Mirror Mode compared to the Standard Mode, but no significant difference was found according to the statistical paired t-test. Refining the model may improve imitation quality and increase player's satisfaction, especially when players face their own strategies.

Keywords: Imitation Learning · Game AI · Strategy Games.

1 Introduction

During the past few years, it has become increasingly apparent that Artificial Intelligence (AI) has taken a prominent role in society. Machine Learning (ML) techniques, such as Deep Neural Networks, have advanced significantly, providing interesting applications of AI across many fields. In Game Development, ML has also become widely studied. In particular, for the enhancement of realistic behaviors of non-playable characters (NPCs).

NPC behavior refers to how characters in games should act and react to certain events in the game environment. Realistic NPC behavior contributes

significantly to the player immersion and satisfaction of the game [17]. Ideally, each NPC should exhibit behavior that aligns with its role in the game. For instance, a knight character might be programmed to act more aggressively in combat, while a cleric would prioritize healing or supporting allies.

Traditionally, these behavior types are handled by Finite State Machines (FSM) or Behavior Trees (BT), where each character follows a set of predefined heuristics and transitions between states based on game events [9]. However, this method of programmable behavior can result in repetitive behavior that makes NPCs predictable in their actions [2][18].

In strategy games, the predictability in enemy tactics can have a major influence on a player's experience. Strategy games require tactical thinking to defeat a team of opposing characters, while keeping your own team alive. Statistics in 2024 have shown that the popularity of strategy games has drastically decreased in the past 9 years [34]. According to this study, the causal reason for this is hard to identify. However, the predictability of the enemy's action can be an important effect to this. Predictable enemy behavior makes the games easier to play, possibly reducing the engagement of a player for more experienced players who recognize the enemy's behavior [2]. In addition to this, it is found that playing several repetitive games can cause boredom [5].

To address the risk of boredom in strategy video games over time among experienced players, this research introduces a new game mode called Mirror Mode, where the enemy NPCs learn a strategy based on the player's strategy by using Imitation Learning (IL)[24]. For this research, a simplified version of the mobile strategy game Fire Emblem Heroes is created. Several Imitation Learning and Reinforcement Learning (RL) techniques are applied to train agents in the Fire Emblem environment and compared to each other, including Generative Adversarial Imitation Learning (GAIL), Behavioral Cloning (BC), and Proximal Policy Optimization (PPO).

The algorithms are tested among different parameters to find the optimal model that is used for the player tests. Ultimately, the optimal hyperparameters are used for the model that are trained on real player's data. Tests are conducted where participants need to play the created game for five rounds against the standard enemy AI. A second test is taken where each player is up against an enemy AI trained by either their demonstrated data or that from another participant.

Their game experience and satisfaction is measured for both the Standard Mode as well as the Mirror Mode via questionnaires and gameplay metrics. Altogether, the collected results from the player tests are used to establish an answer to the following question: "*How will a player's game experience be influenced when NPCs imitate their strategy in a turn-based strategy game?*"

With the results of the finetuning experiments the following sub-question will be discussed: "*To what extent can RL and IL be applied to teach NPCs a player's strategy in a turn-based strategy game?*"

Accordingly, the following hypothesis will be tested: *The game experience of a player will be positively influenced after the enemy AI has imitated their strategy, as the player will be more engaged and more satisfied with the game.*

The remainder of this thesis will discuss the approach and development of addressing these setup questions and hypothesis. First, Section 2 reviews prior research relevant to this study. This is followed by Section 3, which briefly explains the key concepts needed for this research. Before going deeper into the aspects of the study, the setup for the game and virtual environment is given in Section 4.1. After this, Section 4 describes the implementation of the Fire Emblem strategy game variant used as a test environment. Subsequently, the experiments including the corresponding results will be discussed into two sections. Section 5 presents the training and configuration of the imitation models, followed by Section 6 which explains the evaluation of the resulted imitation model through player tests. The results from both experiments will be discussed in Section 7, as well as the limitations of this study and possible future work. Finally, Section 8 concludes the thesis by summarizing the findings and answering the aforementioned questions and hypothesis.

2 Related Work

Before discussing the methods used in this study, it is important to review the background concepts and relevant research related to the topic of this paper. Understanding player engagement essential for motivating the development of more advanced enemy AI. In addition, prior research forms the motivation for the approach conducted in this research. Both topics are discussed in the remainder of this section.

2.1 Game Experience

Game experience is a broad concept in game development and must be implemented effectively to maintain optimal player engagement and satisfaction. Player engagement is strongly influenced by game elements and design choices intended to shape the overall game experience. A foundational theory that guides these design decisions is Csíkszentmihályi's flow theory. According to this flow theory, an individual is most engaged when the challenge of an activity is well aligned to their skill level, creating an optimal state of focus and satisfaction [8][15]. This theory has brought a baseline in game design to maintain a steady balance between the skills of a player and the challenge of the game. To support this balance, many game developers provide adjustable difficulty settings, allowing players to match the challenge to their abilities as they play. However, frequently adjusting the difficulty can break immersion and disrupt the player's sense of flow, making it harder for them to stay engaged over time [15].

Moreover, a player's motivation to play can also shape their experience of the game. Players who focus primarily on achieving in-game objectives tend to

be more satisfied with smooth and accessible gameplay, while those who play for enjoyment are more engaged by challenging and exciting gameplay [6].

These differences in player motivations and preferred levels of challenge, highlight the difficulty of creating a single game experience that satisfies all types of players.

Research by A. Akram et. al, has shown that the player satisfaction has improved after the implementation of AI driven mechanics for the animation of NPCs. They suggest that further improvement on the game satisfaction can include using AI for the adaptability of NPCs on player's gaming behavior [2]. In addition to this, Modern Game AI has already proven its success in record breaking games such as 2020s game of the year, The Last of Us 2, where enemies are more aware of their environment and stay alerted when they found an ally dead [29] [19]. Therefore, implementing new AI methods to improve the adaptability of NPCs and reduce the predictability of their actions, can potentially help maintaining the satisfaction and flow of a game.

However, for a more advanced enemy behavior, different approaches need to be found to create agents that are adaptable to the environment. RL in the context of video games still has its limitations, due to the heavy computational demands, arbitrary reward systems, and slow learning curves. This often makes RL a less desirable method for complex game environments. RPG strategy games are considerably a complex environment, making it challenging to specify a suitable game state for the RL algorithms, and require a high degree of adaptability in the game [35].

2.2 Prior Research

RL and Video Games share a mutual interest, where RL offers new possibilities for creating intelligent game agents, while video games provide a practical environment for testing and developing RL techniques [9]. Over the years, RL has been applied to NPC behavior in games to make them more complex and adaptable.

Early work by Sanchez-Ruiz et al. explored NPC adaptation in the turn-based strategy game Call to Power2, using an ontological approach [28]. They stored previously successful strategies in a case library and organized them hierarchically through Hierarchal Task Networks. When encountered with a new situation, the agent van retrieve actions by looking up similar states in the library. Their Repair-SHOP system makes sure that the plan gets modified of it did not fully suit the current situation, allowing the adaptable behavior. While effective in improving decision-making speed and accuracy, the approach was limited in adapting to entirely new conditions and involved computational complexity due to its ontology-based reasoning.

Researchers from OpenAI demonstrated the adaptability of agents while letting them play a game of hide-and-seek, through self-play [4]. Self-play enables agents to continuously adapt their policy by playing against agents using the same policy that gets updated iteratively. The research applied PPO and Generalized Advantage Estimation to optimize their policy. It was found that through

self-play, agents learned to develop counter-strategies to earlier discovered in-game strategies. Although the hide-and-seek environment differs from turn-based strategy games, this research shows how RL agents can evolve adaptive behaviors based on in-game interactions and strongly motivates the use of RL and self-play for adaptable behavior.

Continuing on this work, OpenAI explored a wider range of possibilities of RL for adaptive behavior. OpenAI Five was the first AI system that was able to defeat the professional players in the multiplayer real-time strategy game Dota 2 [22]. While the hide-and-seek research showed how agents can learn creative new strategies through self-play, the work on Dota 2 takes this a step further by applying RL and self-play to a much larger action and observation space. Their achievement of defeating the world champions in Dota 2, marked a great success for deep RL techniques in stategy games.

C. Amato and G. Shani further investigate adaptive strategy behavior for NPCs, in the turn-based strategy game Civilization [3]. They applied several RL techniques such as Q-learning and Dyna-Q, to make agents learn policies to switch between strategies given the current game situation. The authors represent NPC behavior through a Markov Decision Process. Via RL, the goal is to find a policy π that maximizes the sum of rewards over the steps of the process.

Interestingly, this research focuses on adapting the agents while playing the game in real-time, making the agent behavior unique to a player's current strategy. Their results show that these RL algorithms provide a powerful tool to let agents adapt to game situations as complicated as strategy games. As it shows high potential for the use of RL to teach agents the player's strategy, the authors encourage investigating the use of more advanced RL approaches in the future. This motivates the choice for using IL in strategy games for this research. For these environments, learning with the help of expert gameplay can be more efficient than using standard RL reward systems [35]. By using expert data that demonstrates human gaming behavior, IL offers a new potential ML method for strategy gaming behavior.

One of such IL methods is Generative Adversarial Inverse Learning (GAIL), introduced by J. Ho and S. Ermon [14]. This algorithm makes an appealing method for IL in video games environments. Research by Gharbi and Fennan continue working through this claim by comparing GAIL to other methods that imitate player behavior, such as Proximal Policy Optimization (PPO) and Behavioral Cloning (BC) [13]. They find that GAIL is an effective technique for replicating player behavior in video games, and that they provide a high accuracy in replicating complex player strategies. They suggest that combining methods, such as GAIL with PPO or other model-free reinforcement learning techniques, could be optimal for developing adaptive game AI. Such combinations allow agents to imitate player strategies while remaining responsive to changing game conditions. Their findings are a strong motivation to explore the effect of combining PPO, BC and GAIL, in the strategy video game created for this study.

While no research has yet explored the possibilities of RL techniques to copy player behavior, specifically for turn-based strategy games, these studies together highlight the potential of combining model-free RL with imitation reward systems in video game environments. This combination would allow agents to dynamically mimic player behavior, possibly creating a challenging and engaging game experience. Therefore the possibilities of IL in strategic video games is further explored in this research.

3 Background

To address the possible boredom and decline in popularity in strategy games, this study applies ML techniques to develop a novel game mode. Before discussing the implementation and research setup, relevant background knowledge will be discussed. First, the game rules of the implemented strategy video game will be explained. Subsequently, the ML methods used in this research will be briefly explained.

3.1 Fire Emblem Heroes

Fire Emblem is a turn-based strategy role-playing game developed by Intelligent Systems and published by Nintendo. This research uses a simplified version of the 2017 mobile game adaption, maintaining the complex tactical thinking in a compact 6×8 grid-based map, such as the maps shown in Figure 1.

The game revolves around an alternating player and enemy phase. Starting with the player phase, all units are positioned on one tile on the map. Characteristics corresponding to the player's team can be recognized by its blue color, while the enemy team is always represented by the red color. An example of the start interface can be seen in Figure 1a.

In each phase, all surviving units in a team have one turn. One unit can be selected at a time to perform an action in their turn, which can be to move, wait, or attack. Attacks are only possible if a foe is within the attack range of a unit, and a target may counterattack if their range matches the attacker's range. Figure 1b shows how the game presents tiles that are within attack or movement range. Tiles that are in attack range are highlighted in red, and tiles that are within movement range in blue. A unit can move to any tile highlighted in blue, as given in Figure 1c. If an enemy stands on a tile within attack range, the tile is highlighted in brighter red to indicate that the enemy can be selected to attack. When a target is selected, the combat information appears in the top of the screen, shown in Figure 1d.

The game ends when all units on one side are defeated. Each team contains four units, and the game starts at the player phase.

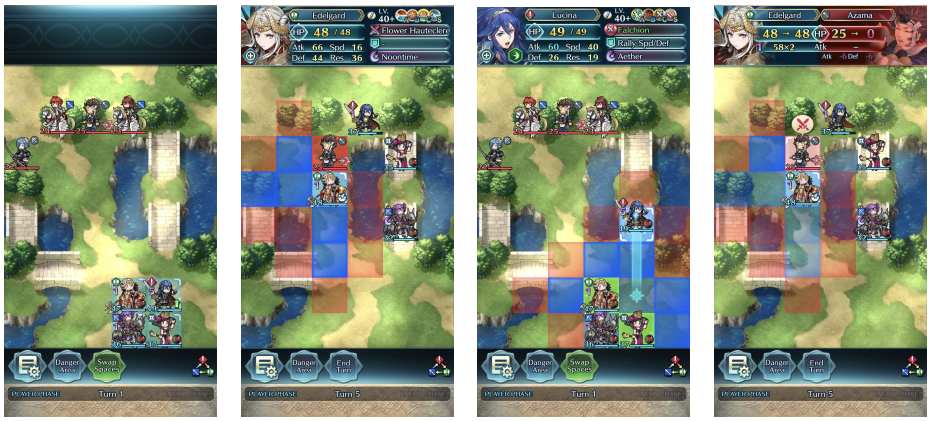
Each unit belongs to one of four types: infantry, cavalry, flying, and heavy armor units. The unit type determines their terrain accessibility, step size, and attack and defense power named as stats. Units also carry one of the five weapon types: sword, lance, axe, bow, or magic. Weapons determine attack range and

can apply advantages and effectiveness. Melee weapons (sword, axe, lance) have a range of one tile, while bow and magic can attack from two tiles away.

The strategic thinking arises from the interaction of unit types, their weapons, and stats. The melee weapons are part of a weapon triangle, presented in Figure 2, similar to the "rock, paper, scissors" principle. Sword has an advantage against axe, axe against lance, and lance against sword. Boosting damage by 1.2x, or 0.8 the other way around.

Bows are highly effective against flying units, dealing 1.5× extra damage. Heavy armor units have a high attack and defense, but a low resistance, making them only vulnerable to magic. Magic units have a high resistance, and are valuable to defeat the heavy armor units, but they are vulnerable to melee attacks.

Lastly, the five core stats are important to keep in mind. HP determines the hit points the unit can take. Attack calculates the damage output by the unit. Defense is subtracted from the damage of an attacker carrying a melee weapon, whereas the resistance is subtracted from the damage of a magic user. The speed enables a follow-up attack when the difference in speed between attacker and target differs at least 5 points.



(a) Player phase start. All units are ready to act while enemies await their turn. Each unit's HP bar appears below its sprite, with a weapon type icon above its head.

(b) The available movement range (blue) and attack range (red) of a selected infantry unit. An enemy that can be attacked stands on a tile highlighted in brighter red.

(c) Movement interface of a selected unit. The unit information is shown at the top. On the map, an arrow indicates the movement to a chosen blue tile within range.

(d) Combat information at the top of the screen. First line shows HP before and after combat. Second line shows attack power × number of attacks.

Fig. 1: An example of a battle map from Fire Emblem Heroes. For a video of a completion of this map, with different characters, see the available gameplay [21]. Units are generally recognized by blue colors, and enemies by red.

Fig. 2: The melee weapon triangle in Fire Emblem, showing the advantage and disadvantage attacks triangle. Sword has its advantage against axe, axe against lance, and lance against sword. The other way around presents the disadvantage chart. This chart forms a key element to the strategic thinking in Fire Emblem.

Originally, the game includes more types than the aforementioned unit and weapon types. In addition, the strategies are widely influences by special abilities, assists, and skills. To maintain simplicity, this implementation solely focuses on the interaction between the four unit types, and five weapon types.

3.2 Proximal Policy Optimization

In Reinforcement Learning, an agent interacts with an environment by observing states, selecting actions, and receiving feedback in the form of rewards or penalties. This is often referred to as extrinsic rewards, as the rewards are defined externally from the algorithm. Rewards, can also be defined as intrinsic rewards, to encourage the agent to explore in certain paths, known as the curiosity rewards [30]. Its objective is to learn a policy π that maximizes the expected cumulative reward over time, given by taken actions a in observed state s. This is usually achieved by estimating value functions that calculate the expected value of a taken action in a given state.

Typical RL algorithms often rely on the *Bellman equation* to iteratively approximate the optimal value function. However, in high-dimensional spaces such as video games, storing and updating value functions for every possible state becomes computationally impractical. Therefore, a Neural Network often comes in as function approximator to estimate the action-value function [20].

A common method is Proximal Policy Optimization (PPO), which improves the stability of policy gradient methods in RL. In PPO, both the policy and the value functions are parameterized using neural networks, allowing learning efficiency in high-dimensional spaces through first-order optimization of a clipped surrogate objective [26], A surrogate objective is used as a replacement to true objective functions that are difficult or impossible to optimize through calculating its derivatives. So a surrogate loss function is used as a proxy that can be optimized to approximate how the true objective would behave when the policy is updated.

PPO directly optimizes a parameterized policy to maximize expected rewards (θ), using a clipped surrogate loss function to ensure stable updates by limiting the degree in which a policy can be changed during an update 1. This stabi-

lization is important, as it prevents the agent from diverging from the optimal solution too quickly. This makes sure there is a maintained balance between exploration and exploitations of the actions.

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

Hyperparameter ϵ determines the clipped range. The first term inside the minimal function is the surrogate objective defined by Trust Region Policy Optimization (TRPO) 2, that functions as a probability ratio between the old and the new policy. The second term modifies this probability by clipping it, making sure too large probabilities are removed. Consequently, the minimal value of the two objectives is taken, and is used to compute a gradient descent to update the policy parameters θ. The degree to which a policy gets updated into the direction of the gradient descent is determined by the learning rate α. A larger value means faster but less stable leaning capabilities, whereas a small value gives slower convergence.

$$r_t(\theta) = \frac{\pi_\theta(a_t \mid s_t)}{\pi_{\theta_{\text{old}}}(a_t \mid s_t)} \tag{2}$$

J. Schulman et al. formulated the methods for PPO and have found that the performance of their method is overall better compared to other optimization policy methods [26]. PPO is a commonly used technique for RL applications.

3.3 Behavioral Cloning

While PPO has demonstrated impressive results in complex environments, a more efficient approach for teaching human behavior to an agent is through human demonstrations. This is where IL becomes useful. Learning from Demonstrations (LfD) falls under IL, which is the approach that trains a policy to mimic the decisions seen in demonstrations. LfD often leads to faster learning and more stable training in early phases [24].

One of the most straightforward approaches to IL is Behavioral Cloning (BC), which uses supervised learning to train a policy directly from a given dataset of state-action pairs given by an expert. Its objective is to minimize the supervised loss function under the state distribution d_{π^*} encountered in the expert data 3. Subsequently, the learning policy $\hat{\pi}_{\text{sup}}$ is obtained by applying standard supervised learning to match the expert's actions with the actions predicted by the policy on the same states [25].

$$\hat{\pi}_{\text{sup}} = \arg\min_{\pi \in \Pi}\ \mathbb{E}_{s \sim d_{\pi^*}} \left[\ell(s, \pi) \right] \tag{3}$$

However, when the learned policy makes a mistake, it might go to a state that the expert has not visited. Once the policy has entered an unfamiliar state, it is more likely to make more mistakes as it has no data for those states. This can increase the computational costs drastically [24]. To avoid this, an immense amount of demonstrations is required, or a combination of BC with other RL approaches. The latter approach is what will be used in this study.

3.4 Generative Adversarial Imitation Learning

Since BC can be inefficient, especially when only a limited number of demonstrations are available, alternative approaches have been found that address this limitation. J. Ho and S. Ermon were inspired by the adversarial methods in Generative Adversarial Networks, and combined these techniques with concepts from IL and Inverse Reinforcement Learning. They proposed a new model-free algorithm that imitates expert player behavior, named Generative Adversarial Imitation Learning (GAIL) [14]. IRL aims to learn the cost function from observed expert data. However, IRL costs a lot of computational power and it does not teach the agent directly how to act as they primarily focus on the quality of the actions. GAIL addresses this by using an adversarial training to teach the agent human behavior without explicitly relying on the reward function.

This training technique is based on a minimax game between a policy π and a discriminator D that is used to distinguish the generated gameplay from the real game play. The policy tries to minimize the objective function by fooling the discriminator by imitating the expert behavior as closely as possible. Meanwhile, the discriminator tries to maximize the objective by getting better at telling the agent and expert apart. This methods creates two distinct loss functions that both value to be minimized. The discriminator loss needs to be maintained properly, a loss very close to zero can cause the vanishing gradient problem, while a loss close to one is caused by a weak discriminator. Therefore aiming for a loss value around 0.5, likely indicates a good generative behavior [33]. Eventually, the aim is to retrieve a saddle point (π, D) of the expression defined in equation 4, where the policy has learned to imitate the expert at a level that the discriminator cannot tell the difference. The influence of this imitation signal from the discriminator is scaled by a strength parameter, which controls how strongly the discriminator's estimations shape the policy updates during training.

$$\mathbb{E}_{\pi}\left[\log D(s,a)\right] + \mathbb{E}_{\pi_E}\left[\log(1 - D(s,a))\right] - \lambda H(\pi) \tag{4}$$

The last term in the objective, $H(\pi)$ represents the entropy of the policy. This encourages exploration by preventing the policy from becoming fairly deterministic early on in the process, ensuring that the agent continues to try diverse actions. The coefficient λ balances the trade-off between imitating the expert and maintaining sufficient randomness in the agent's decisions.

As GAIL trains agents to closely follow expert trajectories without requiring explicit reward signals, it is ideal for scenarios where expert data is available but defining rewards is challenging such as game environments. Ho and Ermon found that GAIL is effective at generating complex player strategies, making a progress for imitation learning in games.

Component	Version
Unity Editor	2023.2.13f1
ML-Agents Toolkit	3.0.0 (Release 22)
Python	3.10.11
PyTorch	2.2.1
Pip	25.0.1
Operating System	Windows 11

Table 1: Software Versions Used for Environment Setup

4 System Design and Implementation: Fire Emblem in Unity

To investigate the impact of an enemy AI that imitates player behavior, a game environment is developed in Unity Game Engine [27]. The process of this development is detailed in this section. First, the backend and application setup required for the system are reviewed in Section 4.1. A short review of the scene setup presenting the Fire Emblem elements in Unity, is given in Subsection 4.2. Subsequently, the implementation of the game environment and agent script is explained in two separate subsections: Standard Mode 4.3 and Mirror Mode 4.4.

4.1 Environment and Technical Setup

A 2D game environment is created in Unity Engine. For the integration of ML tools from Python PyTorch library with Unity, the ML-Agent toolkit has been developed by A. Juliani et al., including the RL and IL algorithms programmed in Python [16]. This package is available in Unity asset store, and is installed in the created game environment for the research. The installation steps and setup instructions are followed according to ML-Agent version release 22 [1].

After the installation of ML-Agents in the Unity game environment is finished, a virtual environment needs to be set up as an interaction tool between the game environment and the ML tools in Python. A virtual environment is set up in the directory of the game environment project, through Command Prompt in Windows. Once the new virtual environment is activated, Python, ML-Agents, and PyTorch can be installed within the environment.

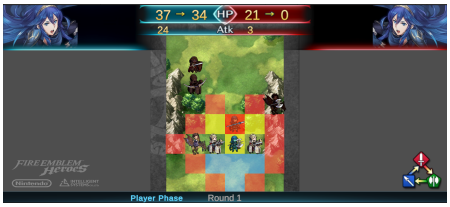
For compatibility between Unity's ML-Agent package and PyTorch tools, it is necessary to acquire the specific application versions listed in table 1.

For a more detailed description of setting up the game environment and virtual environment, a step by step instruction list is added the Appendix Section A.1 and A.2.

4.2 Fire Emblem Unity Scene

The game environment is inspired by the tactical gameplay of the Fire Emblem series. It is built as a 2D grid-based map, from *6x8* tiles, designed with complete

(a) Interface displayed when selected a unit. Movement range (blue) and attack range (red) are highlighted. The information UI is shown in the top left corner, presenting the stats of the selected unit.



(b) The combat interface displayed when a combat is initiated. The left side shows the unit's health before and after the combat, while the right side shows the same for the enemy.

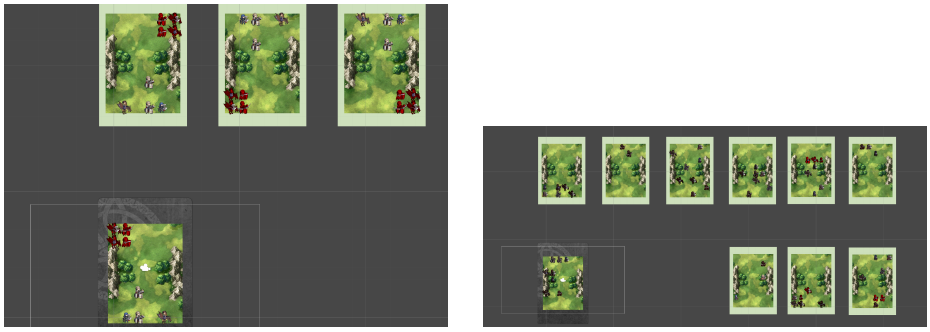
Fig. 3: End result of the implemented version of Fire Emblem Heroes for this study. The game is developed in Unity Engine, and the sprites are collected from the original game sourced by the Fandom page [12]. Player units are presented in their original colors, whereas enemy units are shown in darker red shades.

symmetry across both axes. Each team starts on its own side of the map, ensuring that all tile types and movement distances are balanced and mirrored between the player and enemy units.

Units are defined by a combination of movement type, weapon type, and combat stats. These variables are set public and can be assigned in the Unity Inspector tab, in the Information component of the unit. The enemy types are completely randomized each game round. For both teams, the combat stats are randomized. Standard Mode and Mirror Mode are each created in a separate scene. In both scenes, all visual sprites and icons are collected from publicly available Fire Emblem Fandom community resources in the game assets page [12], and the unit sprites from the character misc information from the heroes list [10]. These are used solely for academic and non-commercial purposes. The resulting interface of the implemented game is shown in Figure 3, with the given interface after selecting a unit in Figure 3a and initiating a combat in Figure 3b.

4.3 Standard Mode Environment

The standard scene includes the general gameplay, and is used to collect playing data. In order to collect the necessary amount of data to train the agent model, the scene includes multiple augmented versions of the map that mirror the player's actions across different axes, presented in Figure 4a. These augmented environments are created by flipping the original map configuration along the x-axis, y-axis, and both axes, resulting in four distinct versions of the map. In the unity scene, each environment manager component should be assigned to the Array of environment managers in the *AugmentationManager* component. The first element in this array is always the original environment where the player acts. The remaining three are enlisted as augmented environments, and are used



(a) The four mirrored environments used to collect player data. Each move set in the original environment, is copied in its mirrored orientation.

(b) The ten environments used for training the agent models. All environments run in parallel, each controlling and playing their own game.

Fig. 4: Environmental setup for collecting data and training agents. The original environment used for collecting and transferring actions is seen in the lower left corner, surrounded by a white frame.

to mirror the player's actions in real time. This allows a single set of demonstrations to generate four unique moves at one time step. The ML-Agents package provides the *Demonstration Recorder* component, which is added to each player unit to record the actions provided by that specific unit. To start recording, the Game Scene needs to be activated and the player start the game. All the actions are automatically recorded, and stored in demonstration files.

In addition, six more environments are included to further accelerate training. These environments are direct copies of the original environment and do not operate as augmented environments. In the inspector tab, all environments used for training should be added to the *Environments* list in *AugmentationManager*. The augmented environments are also used as independent environments during training, and do not function as augmentations of the original environment in this case. These environments operate in parallel when training is enabled. The ten environments run in parallel during training are presented in Figure 4b. To train the model based on IL algorithms, the demonstration folder directory must be specified in the *demo_path* in the configuration file that is used for training. To begin training, the virtual environment from section 4.1 has to be activated in the command prompt. Training can be activated when using the following command: **mlagents-learn** *directoryfilename.yaml –run-id=ID-Name-Run*. Once the game scene is started in Unity, the training progression begins.

Standard Enemy Behavior Script. For the Standard Mode, a rule-based enemy AI is implemented to emulate enemy behavior in the original Fire Emblem Heroes game. The standard enemy tactics were derived through gameplay observation by expert players, and documented by Game8 Inc. [11]. According to

their analyses, the standard enemy AI follows a set of prioritized decision rules. Most importantly, enemies always attack if there is an opposing unit within their attack range. When multiple targets are in range, the enemy chooses the unit to which it can deal the most damage. If no opposing units are within attack range, the enemy holds its position. Furthermore, the order in which enemy units take their turns is determined by the following a set of priorities. Melee attackers are prioritized over ranged attackers, and are allowed to attack first. Among units with the same attack range, it prioritizes the units that can reach the player units the fastest. If all factors are equal, the leftmost unit in the map acts first.

Player Agent Behavior Script. The player's actions need to be processed and saved through the ML-Agents package. This package provides all necessary functions and connections for collecting the player's behavior, and replicating it. A script is created to handle player's action, and is given to all player units. The script uses three main functions offered by the package, to collect observations, mask action spaces, and receive an action.

The first function, *CollectObservations(VectorSensor sensor)*, analyzes all of the observations concerning the current state of the map based on the agent's local position. These observations are important for training the neural network that is used for the decision making when an action needs to be chosen. The state of the game is constructed by unit's and enemy's stats, weapon, and unit types, and the manhattan tile distance between the unit and an opponent. For more compatible computation all values are normalized to a value between 0-1.

After observing the current state, invalid actions can be masked from the model, to prevent these actions from getting selected. This is provided by *WriteDiscreteActionMask(IDiscreteActionMask actionMask)*. For this study, the function filters out the tiles that are not reachable for an agent at a current state, and the targets that are not available to attack.

Once the state has been observed and the invalid actions have been masked, the agent receives an action processed through *OnActionReceived(ActionBuffer actions)*. The method through which the agent receives actions, can be decided through a heuristics only behavior type or a default behavior type. Before playing Standard Mode, the *Behavior Type* in each unit's *Behavior Parameters* component must be set to *Heuristic Only*, allowing manual player input to control the unit. Additionally, all units should share the same *Behavior Name*, to ensure that their actions are all sent to the same demonstration.

Training models occurs in the Standard scene as well, and the *Behavior Type* should be switched to *Default*. This allows the agent to request decisions from the learned policy, and update it after received rewards.

The actions that are processed and received are identified by three discrete action variables: action type, selected tile, targeted unit. An action type holds an index that corresponds to the action chosen by the player, which can be to wait (0), to move (1), or to attack (2).

The selected tile corresponds to the index of a tile within the map size *6x8*, resulting in an array of size 48. The selected tile holds the index of the tile that

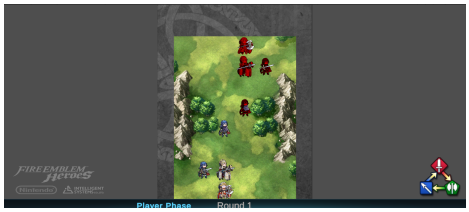


Fig. 5: Mirror Mode in the implemented Fire Emblem environment for this study. The player's team and enemy's team are completely mirrored from each other, in positioning and typing.

the unit has selected to move to. For the waiting action, this equals the index of the tile that the unit is currently already on. For the attack action, the tile that the unit stands on to launch it attack from is used.

Lastly, the targeted unit decides the target that will be attacked during an initiated combat. If no combat is initiated, the value is of no use. In case an attack is performed, the index of the target unit is used. Therefore, in order for an attack action to be valid, the selected tile and the targeted unit need to be correctly chosen.

4.4 Mirror Mode Scene

Mirror Mode includes only one environment, and is used to evaluate the agent model trained on player demonstrations. The enemy team is a complete mirror of the player's team in terms of positions, unit types, and weapons. The units only differ in stats. An example of a Mirror Mode state is presented in Figure 5. Compared to Standard Mode, the main difference lies in enemy behavior. Where Standard Mode used a traditional heuristic approach, the enemy AI in Mirror Mode uses the models trained by RL and IL algorithms.

Mirrored Enemy Behavior. Similar to the unit agent function discussed in subsection 4.3, each enemy unit holds an agent script and behavior script. The enemy agent script is identical to player agent script, with a few differences in variable names, making variables specific to the enemy. This allows clear identification between a player and an enemy agent. In addition to this, unlike the player agent, an enemy agent does not receive actions from player input. The action decision process is purely controlled by the learned algorithm, provided by trained model. When it is the enemy's turn to take an action, it requests a decision from the trained model submitted in the model parameter from the *Behavior Parameters* component. After requesting an action, the used model outputs the discrete actions array based on the collected observation. The enemy agent uses the array to set the actions. Similar to the mechanics in the unit agent, the first value in the array corresponds to the action type, the second value to the selected tile index to move to, and the third value to the targeted unit to

attack. If an enemy has decided to initiate a combat, the selected target needs to be within attack range of the selected enemy, from the selected tile index. If the selected tile does not allow the enemy to attack the target but the target can be attacked from another tile, the enemy can pick one of the valid tiles randomly. This allows the enemy AI to attack even when the model returned the wrong tile index, to increase the offensive behavior style.

Player Unit Script. The player unit interaction is handled the same way as in the Standard Mode, however excluding the recorded demonstration functions and the functions that transfer the actions to other environments. The input for action handling is completely controlled by the player's input, setting the behavior type of each player unit to *Heuristics Only*, in the *Behavior Parameters* component.

4.5 Agent Setup

For the RL algorithms, a practical reward system and learning conditions had to be found. Throughout the process, several tests have been done to find a setup that worked optimally for teaching the agents the right playing behavior.

First rounds of testing were unsuccessful as the agents failed to move. They repeatedly selected invalid actions, causing them to remain idle. To address this, the reward system was adjusted to the recommended range of *[-1,1]*, which led to small but immediate improvements. After a few episodes of training, the agents began to move between tiles. However, the agents still did not perform any attack actions correctly, often targeting opponents out of reach or attempted attacks from invalid positions.

Interestingly, after an hour of training, flying agents learned to move to the lower left corner of the map and stay there, presumable because this position was out of reach for ground-based units. Figure 6 present this case. Additional randomization was introduced to the training process to improve the agents' adaptability. In addition to randomizing starting positions for each episode, the agents' side of the playing field was also varied. This was an important adjustment, making the flying agents less attracted to the corners.

To reduce the learning space for each observation state, the masked actions function discussed in section 4.3 was enabled. This increased the learning speed, and degree in valid actions.

Eventually, killing enemies and winning a round was rewarded by +1, and a penalty of -1 was given to dying or losing. Any other rewards for choosing valid actions received a reward of +0.3.

Again after an hour of training, the agent started to repeatedly jump from one tile to another, without any attempt of attacking the enemy. To prevent this, a maximum set of actions per game was implemented. After the maximum actions is reached, the game ends in a tie resulting in a punishment of -1. This improved the learning curve, making sure the agents did not avoid attacking targets.

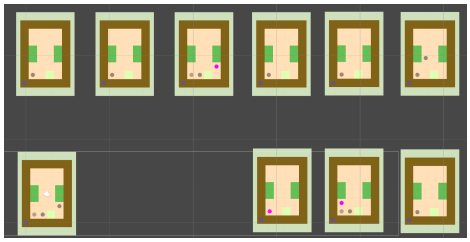


Fig. 6: Early stage of the developed strategy game. The agents found a tactic to move to the lower left corner and stay there to protect themselves.

Finding these settings, the agents were fully ready to learn from the environment and demonstrations, with no further complications that lead to unjustified behavior. The rest of the research continued with these agent settings.

5 Agent Training and Testing

For investigating the possibilities of ML algorithms in the adaption of strategy behavior, the agent models need to be trained according to a set of optimal hyperparameters. Moreover, the optimal model needs to be found through a series of additional experiments where several combinations of ML algorithms are tested. These two experiments form the first half of the experimental setup for this research paper, to find an answer to the question whether RL and IL can be used to imitate player's strategy in video games.

5.1 Experiment 1: Hyperparameter Optimization

The aim of the first experiment was to optimize the performance in cumulative reward, by adjusting one parameter at the time, while keeping the others constant. The value range of each parameter recommended by the ML-Agent developers were taken into consideration [31]. With a duration of over a month, these optimizing experiments were held. The parameter values that were used for optimization are summarized in table 2, with other parameters set to their default values.

Optimizing the models for this study purely focused on the following hyperparameters:

- BC strength;
- PPO learning rate α;
- GAIL learning rate α;
- extrinsic strength;
- curiosity strength.

For identifying the best learning rate for PPO and GAIL, and the best strength for BC, no extrinsic rewards or curiosity parameters were used. The

Tuned	**PPO α**	**GAIL α**	**BC str**	**Curiosity str**	**Extrinsic str**
PPO α	**0.0005, 0.0003, 0.0001**	0.0001	0.0	0.0	0.0
GAIL α	0.0003	**0.001, 0.0005, 0.0003, 0.0001**	0.0	0.0	0.0
BC str	0.0003	0.0001	**0.4, 0.5, 0.6, 0.8, 1.0**	0.0	0.0
Curiosity str	0.0003	0.0001	0.4	**0.0, 0.05, 0.1**	0.0
Extrinsic str	0.0003	0.0001	0.4	0.1	**0.1, 0.5, 1.0**

Table 2: Hyperparameter tuning overview. The diagonals show the values for the hyperparameters that are tested. Other values are fixed and are used for testing the hyperparameter in the corresponding row.

goal was to find suitable values for these parameters to stimulate learning by keeping track of the cumulative reward over time, but also to encourage imitating player demonstrations by maintaining a proper GAIL discriminator loss around a value of 0.5.

5.2 Experiment 2: Model Configurations

After identifying the optimal performing model from Experiment 5.1, further experiments were conducted to determine the optimal configuration for the enemy AI in Mirror Mode. Several combinations of RL and IL techniques were tested.

The following model variants were evaluated:

- PPO only
- PPO + GAIL
- PPO + GAIL + BC
- PPO + GAIL + BC + Curiosity;
- PPO + GAIL + BC + Curiosity + Extrinsic Rewards;
- PPO + GAIL + BC + Extrinsic Rewards;
- PPO + GAIL + BC + Extrinsic Rewards + self-play

Each model was trained for 200,000 steps starting at step 0 with no prior knowledge yet. Models were evaluated based on the cumulative reward and GAIL loss. Similar to the first experiment, the cumulative reward served as a measurement for leaning capability of the agents, and GAIL loss for the imitating abilities of player behavior.

During this phase, a fixed set of hyperparameters was used for consistency across models, as listed in table 3.

Parameter	Value
PPO learning rate	0.0003
PPO hidden units	256
PPO batch size	128
GAIL learning rate	0.0001
GAIL hidden units	64
GAIL gamma	0.85
GAIL strength	1.0
BC strength	0.5
Extrinsic strength	0.9

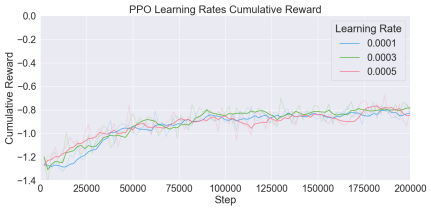
Table 3: Hyperparameters used during model combination testing.

5.3 Results

The first part of this study looked into the possible hyperparameters and configurations for the model that will be applied to the Mirror Mode agents. Several tests were conducted to find a model combination most suitable for the agent behavior to imitate player strategies, as previously discussed in subsections 5.1 and 5.2. This subsection discusses the resulted model hyperparameters and configurations.

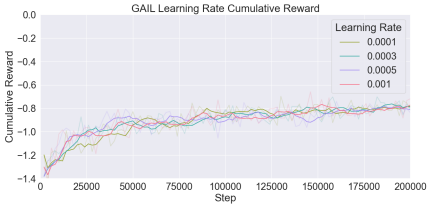
Model Finetuning The first experiment tested several hyperparameters from different RL and IL techniques, mentioned in Table 2.

The figures presented in Figure 7 show the results of finetuning the learning rate values α for PPO and GAIL models, and the strength of BC. The results indicate a gradual learning progression for both PPO and GAIL, whereas BC shows a flattened learning curve. PPO and GAIL converge toward a relatively high cumulative reward of approximately -0.8, with GAIL demonstrating a faster convergence. For learning rates $\alpha = 0.0003$, $\alpha = 0.0005$ and $\alpha = 0.001$, GAIL reaches a cumulative reward of -1.0 after roughly 25k steps, while PPO achieves this milestone after roughly 40k steps. The absence of environment interaction in BC may explain the flat learning trend. Despite the limited learning progression, the BC model achieves a rather good cumulative reward compared to the PPO and GAIL finetuning models. Specifically, a BC strength of 0.4 starts below -0.8, and improves to a higher reward of nearly -0.6, ultimately outperforming the PPO and GAIL. PPO results in a much higher GAIL discriminator loss, of roughly 0.9, compared to other models, as seen in Figure 7b, suggesting a poor performance of the GAIL discriminator. Introducing a higher value for GAIL α causes the discriminator loss to drop. Figure 7d shows a much wider range in discriminator loss over the training steps for GAIL compared to other finetuning models. However, maintaining a balanced discriminator loss is crucial, as excessively low values can cause gradient vanishing, hindering imitation learning. BC further reduces the loss, often pushing it too close to zero. Higher BC strength values amplify this effect, causing a more drastic decrease in the loss.



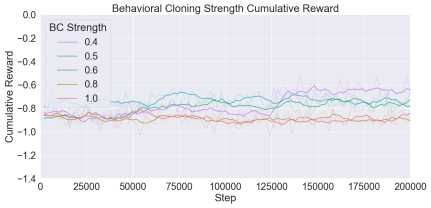
(a) The cumulative reward for several PPO α values. All curves incline gradually to an optimal cumulative reward of -0.8. α=0.0003 gives a slightly better performance, and steadier growth. The results confirm good learning, but not high optimal rewards.

(b) The GAIL discriminator loss of PPO α tuning. The loss declines toward 0.9 across all values α, with α=0.0005 resulting in a slightly lower loss overall, and α=0.0003 showing a slower convergence. A discriminator loss of 0.9 is rather high, indicating a poor discriminator quality.



(c) Cumulative reward of tuning GAIL α. A gradual growth toward a reward of -0.8 is noted, with α = 0.0003 exhibiting a more stable curve. The results further confirm a consistent learning progress.

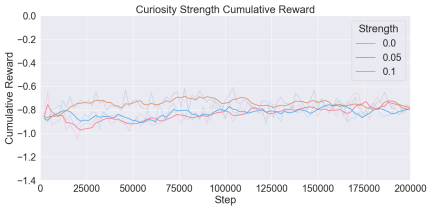
(d) The GAIL discriminator loss of different GAIL α configurations. Larger values for α give a more rapid decrease in loss, and converge to an overall lower loss.



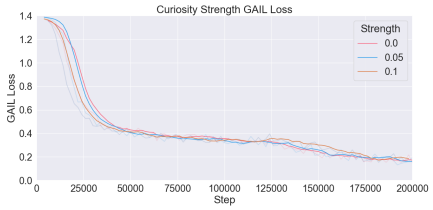
(e) The cumulative reward of different BC strength values. A larger strength results in a lower cumulative reward. The curves show little progress. As BC is not focused on environmental rewards, the learning does not show any growth. A small increasing slope is present after 125k steps for strength=0.4, resulting in the highest reward across the five configurations.

(f) The GAIL discriminator loss for several strength values for BC. Larger strength values decrease the discriminator loss, converging to a value of nearly 0.1. Strength of 0.4 and 0.5 are less steady but converge to a slightly larger loss. The irregular peaks present in str=0.4 and str=0.5 are possibly caused by interruptions of the model training process.

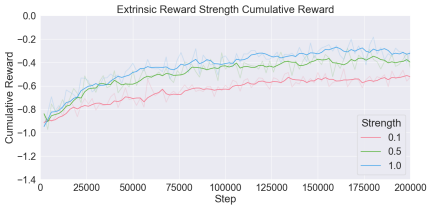
Fig. 7: Results given by finetuning PPO α, GAIL α, and BC strength, over 200k steps. All curves are smoothed by a factor of 5, the original curves are transparent in the background. It is aimed to achieve a high cumulative reward, with a maximum possible value of 2.0, and a GAIL discriminator loss around 0.5.



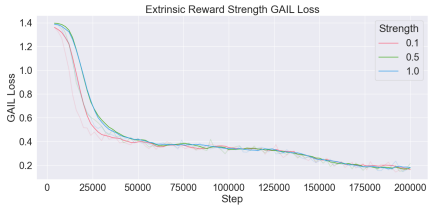
(a) The cumulative reward for different curiosity strength values. Curiosity 0.1 seemingly outperforms the other values. All values converge to a similar cumulative reward level, while a strength set to 0.1 appears to be give a slightly higher mean reward.



(b) The GAIL discriminator loss set for several curiosity strengths. There seems to be little difference in loss when curiosity is applied, compared to when the curiosity reward is disabled. Overall, adding curiosity seems to drop the loss more rapidly.



(c) Cumulative reward given by different extrinsic strength configurations. An immediate improvement in learning can be observed, with strength=1.0 achieving the highest reward. Thus extrinsic rewards highly benefit learning abilities.



(d) GAIL discriminator loss set for extrinsic reward strengths. The loss decreases rapidly, with strength=0.1 declining more quickly than the others. Showing that extrinsic rewards possibly negatively impacts agents imitation abilities.

Fig. 8: Results given by finetuning strength values for curiosity and extrinsic rewards. All curves are smoothed by a factor of 5, the original curves are transparent in the background. A higher cumulative reward indicates better agent performance, with a maximum possible reward of 2.0. The GAIL discriminator loss should be around a value of 0.5. Overall, curiosity shows little effect to the results, whereas extrinsic rewards rapidly increases cumulative reward and decreases GAIL loss.

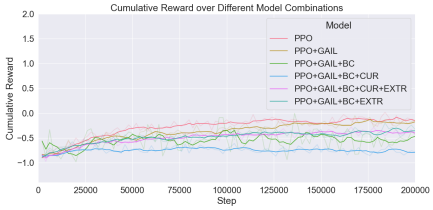
In addition, extrinsic and curiosity rewards are added and tested among different strength values, presented in Figure 8. Interestingly, introducing an extrinsic reward accelerates the learning curve drastically, with the cumulative reward converging to nearly -0.2 when the strength parameter is set to 1.0. However, extrinsic reward reduces the GAIL discriminator loss, dropping too close to zero, which may indicate poor imitation quality. In contrast, incorporating a curiosity-based reward appears to have little to no impact on either the cumulative reward or GAIL loss. The results closely resemble those from BC

finetuning models without curiosity, suggesting limited effectiveness of curiosity in this setup.

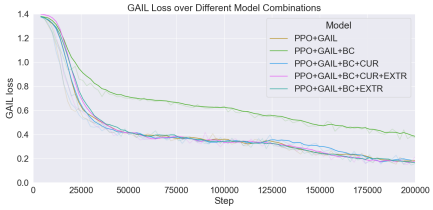
Overall, the results lead to a clear indication of learning capabilities of the agents. A trade-off between imitation ability and cumulative reward performance needs to be maintained. Extrinsic rewards positively affect the cumulative reward, leading to better performing agents. However, dropping the discriminator loss for GAIL, suggesting less suitable imitation behavior. BC and GAIL provide better imitation behavior, but perform less well as they rely less on interacting with the environment. Considering all this, it was chosen to continue with a PPO learning rate set to $\alpha = 0.0003$, and GAIL $\alpha = 0.0001$. Moreover, BC was retained to enhance the imitation learning, with its strength set to 0.4. Curiosity was excluded due to its minimal impact, while extrinsic reward was set to a moderate value of 0.5 to maintain the balance between imitation and exploration.

Model Configurations The second experiment involved different variations of RL and IL techniques to find a suitable model for imitating player strategies. The different combinations are assessed on cumulative reward for overall performance, and GAIL loss as an indicator for imitation performance. The results are presented in Figure 9.

As a continuation on the second experiment, alternative training techniques are applied to see any potential effects on performance and imitation behavior. As the GAIL loss in the different model configurations presented in Figure 9b, tends to drop quickly toward zero, it is expected that the models will imitate



(a) Cumulative Rewards for different ML combinations. All models provide a steady learning curve, indicating good learning progression. PPO+GAIL+BC provides a less steady curve, indicating less certain learning behavior, whereas curiosity seemingly flattens the curve. PPO and GAIL result in the highest reward, giving the best performance.



(b) GAIL discriminator loss for different model configurations. The loss for most models decreases rapidly toward a value of 0.2, whereas PPO+GAIL+BC declines more gradually. None of the models are too close to zero, possibly suitable for imitation capabilities. PPO model is not present in the figure, since it does not use GAIL, suggesting no imitation behavior.

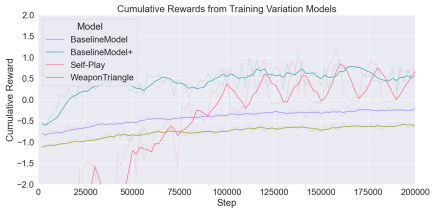
Fig. 9: The results for different RL and IL combinations, over 200k steps. Each model is tested with the parameters found in the experiment 5.1. The curves are smoothed by a factor of 5, and the original curve is present in the background.

the player behavior less well over a longer period of steps. Moreover, the final cumulative reward, shown in Figure 9a, still fails to reach any positive values. As the maximum reward value equals 2.0, the agents still need a lot of progress.

Final alternative training adjustments are made to improve this behavior, presented in Figure 10. One model is trained by using self-play. Another model is trained by reducing the number of weapons used in the game, solely focusing on melee weapons from the weapon triangle. The last model is referred to as the Mirror Model, which has an increased PPO batch size of 256, and 128 hidden units for GAIL, to ensure slower GAIL loss convergence.

The three new models are compared to the model trained with the optimal hyperparameters found in experiment 5.1, as the baseline model. According to the results, the new batch size and hidden units show an quick improvement in cumulative reward as well as a much slower convergence in GAIL loss. The self-play model shows a promising growth in cumulative reward, going from a large negative value to positive reward. However, unlike the Mirror Model, self-play quickly decreases in GAIL Loss, reaching a value too close to zero, making it not a reliable model for imitation purposes.

Training a model purely on weapon triangle weapons shows no better results than the baseline model with the optimal parameters, indicating that fewer weapon types does not improve agent performance.



(a) Cumulative rewards across model variants. Self-play shows the steepest improvement, rising from below -2.0 to around 0.5. BaselineModel+ converges quickly to 0.6 despite a less stable curve. Reducing the weapon space lowers performance, falling behind the baseline. Overall, BaselineModel+ and self-play demonstrate strong potential agent behavior.

(b) GAIL discriminator loss over different model training adjustments. The loss has decreased more rapidly for self-play and weapon triangle, suggesting less reliable imitation behavior. The baseline+ model seems to slow down the loss over time less rapidly, making it more suitable for training the Mirror Mode models over a larger period of steps.

Fig. 10: Results of additional model adjustments aimed at reducing GAIL loss and improving cumulative reward. Curves are smoothed (factor 5), with original data shown transparently in the background. The baseline is the optimal configuration from Experiment 5.2. BaselineModel+ increases the PPO batch size to 256 and GAIL hidden units to 128. Self-Play applies the baseline in a self-play setup. Weapon Triangle limits attacks to melee weapons, reducing the action space.

Taking the results of the first and second experiment all together, it was chosen to continue the player tests with the Mirror Model architecture computed in Experiment5.2 and the hyperparameter setup found in Experiment 5.1. The configuration of this model is added to the Appendix A.3. This configuration file is used for the third experiment, where the models are trained based on real participant demonstrations.

6 User Tests: Mirror Mode Player Evaluation

The model configuration found in the first two experiments discussed in Section 5, is used to train the enemy agent models for the Mirror Mode. Player tests were conducted to evaluate the effect of the Mirror Mode on player experience. To evaluate this, 12 participants were found to compare their playing behavior in Standard Mode to their behavior in Mirror Mode. Participants were divided into two groups, an experimental group and a control group, each with six players. This section discusses the experimental setup of the player tests, and the collected results.

6.1 Experiment 3: Player Tests

Both qualitative as well as quantitative data is collected to evaluate effect of the Mirror Mode algorithm on a player's gaming experience and satisfaction. It is inspired by the methods used by Pagulayan et al. [23], which combined qualitative, quantitative, and observational data for assessing user satisfaction.

Only players experienced with strategy video games were asked to participate in this research, such as those familiar with the Fire Emblem series. Players were asked beforehand to rate their skills and familiarity with turn-based strategy video games.

In total, the study included 12 participants, randomly assigned to a control group ($n = 6$) or an experimental group ($n = 6$). The experimental group played against their personal strategy in the Mirror Mode, whereas the control group played against the strategy of another player from the experimental group. Each participant took part in two test sessions conducted on-site. In order for the AI model to learn from a participant's gameplay, the two game modes for the test group were tested on separate days.

On the first day, the participant played the Standard Mode for five full rounds. During this session, their game state and action pairs were recorded. Additional game behavior metrics were tracked in the agent script, such as attack advantages and disadvantages, total movements, and total attacks applied. After the playing session, a survey was administered to assess the participant's experience and satisfaction. The survey provided a satisfaction score, with a maximum possible score of 45 points, indicating a player's satisfaction of the game.

On the second day, the participant played the Mirror Mode, allowing the AI to utilize the learned model from the Standard Mode. Similar playing metrics

were collected, but this time from the enemy agent using the Mirror Model. The session was again followed by the same survey for the satisfaction score, with additional questions that focus on comparing the Standard Mode and the Mirror Mode.

Each test was concluded with a few additional questions about the player's experience, recorded in the survey. This casual interview allowed participants to provide qualitative feedback about their overall experience across both modes. Interview questions recorded the game experience, interests, and skills of the participants. Moreover, they were asked to describe the enemy behavior and any potential differences that they noticed between the two game modes.

6.2 Results

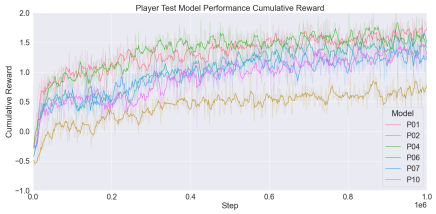
The results from the third experiment were collected through questionnaires, filled in by each participant. Only players familiar to strategy video games took part in this research. They were asked to rate their skills and familiarity beforehand, with the results presented in the Appendix 16. Independent t-test showed no significant differences between the experimental and control group in familiarity $t(10) = 0.00, p = 1.00$, or in skill level $t(10) = 0.42, p = .69$. Hence, the groups were considered fairly divided with respect to prior experience and knowledge.

The game experience was measured through 9 rating questions on a scale from 1-5, to compute a satisfaction score. The questionnaire also served as a platform to record participant's answers to the interview questions, to find more details about their experience and potential remarks about the new enemy AI.

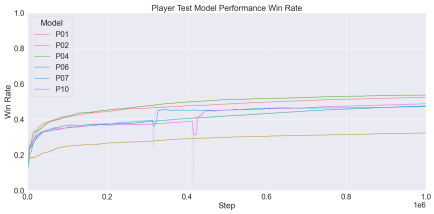
In addition, the participant's gameplay results were tracked and compared to the models trained on the demonstrations to the corresponding participant of the experimental group.

Player Metrics Each game in Standard Mode played by a participant provided demonstrations with their actions to corresponding observed states. Models are trained uniquely to the demonstrations provided by participants in the experimental group, resulting in six distinct agent models. The trained models are assessed by the overall performance in cumulative rewards and win rate across $1million$ training steps, resulting in the learning curves observed in Figures 11a and 11b. In the figures, the models are named after the participant ID that was used to train the model. A difference in performance can be noted across models. The model of P01 and the model of P04 perform similarly well, converging to a cumulative reward of over 1.5. Both models end with a high win rate, of nearly 60%, making them the best performing models out of the models from the experimental group. P02 performs the least well, converging to a reward below 1.0, with a win rate of nearly 40%.

In Mirror Mode, participants in the experimental group played against enemy agents that used a model trained on their own demonstrations collected during their Standard Mode gameplay. The performance of the agent models



(a) The cumulative rewards of models trained on participant demonstrations. It can be observed that there is a difference in performance across models, with $P02$ performing the least, and $P01$ and $P04$ the best. All models gradually converge to a high cumulative reward over 1. With $P01$ and $P04$ nearly encountering the optimal reward of 2.0, suggesting their models result in the best performance.



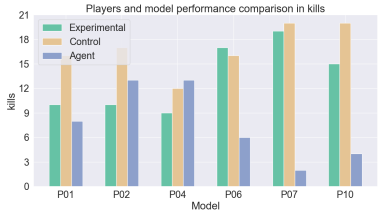
(b) Win rate across different participant models. Win rate equals total won games over the total games played over all episodes ran. All models steadily converge to a higher win rate. The small dips present in $P07$ and $P10$ are caused by an interruption during training. Results suggest that $P01$ and $P04$ perform the best by encountering a win rate of nearly 60%. $P02$ performs the least.

Fig. 11: The results of the six player models trained on participant's playing demonstrations. The models are trained over 1M steps, with PPO α=0.0003, GAIL α=0.0001, BC strength=0.4, no curiosity, and extrinsic strength=0.5. The curves are smoothed by a factor of 5, the original curves are presented in the background in transparent undertones.

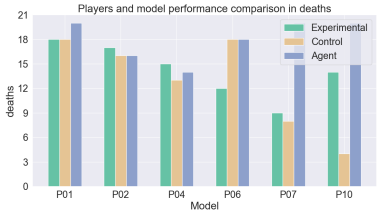
is measured during Mirror Mode gameplay of each participant. The metrics of each model are taken over a total of 10 rounds. Five rounds are against its corresponding experimental participant, and five against one control participant. This way, each agent model is tested twice. The performance is based on eight in-game metrics. The total measured score over the 10 rounds is divided by two, to calculate the average performance of the agent for each participant the agent competed against. The metrics taken contain the total number of defeated opponents after five rounds is measured and shown in Figure 12a, and the number of fallen units over a total of five rounds is presented in Figure 12b. The total number of times an attack or movement is applied are given in Figures 12c and 12d. Additionally, attacks that had advantage, disadvantage, or effect, were tracked and are shown in Figures 13a, 13b, and 13c respectively. In order to evaluate the imitation quality of the agents, participant performance is measured as well of each participant during the five rounds of Standard Mode gameplay. Agent performance is compared with that of the participants they competed against to assess how well agents imitate. Each metric can compared between the participants from the experimental group, the participants from the control group, and the agents. The results are presented in Figures 12 and 13. High similarity between the agent and experimental metrics, together with low similarity to the control-group metrics, indicates strong imitation quality. Similar metric values

between the control and agent bar indicate that it cannot be assured that the agent imitates the player presented by the experimental bar.

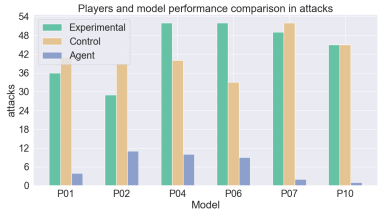
The results indicate that Mirror Mode models struggle to imitate offensive behavior. The agents rarely perform attacks and fail to replicate effective and advantage attacks, compared to their corresponding participants. However, Figure 12d shows that the agents closely resemble the movement patterns from the experimental players they were trained on. In particular, agents trained on P01, P02, and P10 demonstrate a strong alignment with their player behavior. In con-



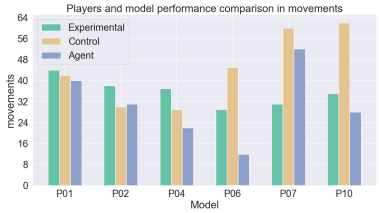
(a) Number of defeated units in the opposing team. Results show no similarity between player and agent behavior in regards of kill count.



(b) Total number of fallen units in the corresponding team. The results show some similarity between player and agent behavior in respect to death rate.

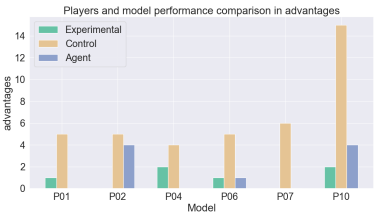


(c) Number of attacks performed during gameplay, by the corresponding team. The results show very different attacking behavior between players and agents.

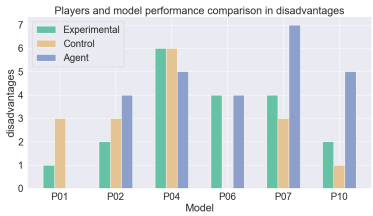


(d) Total number of movements made by the corresponding team. The results indicate some similarity in movement behavior between player and agent.

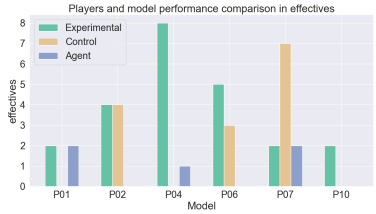
Fig. 12: Comparison of agent and participant performance across in-game metrics. Agent bar shows the performance of the enemy agents in Mirror Mode that were trained on demonstrations from participants in the experimental group. Agent performance was averaged over ten gameplays: five vs. experimental and five vs. control participants. In total six models were trained, one for each participant from the experimental group with $N = 6$. The experimental bar shows the performance of the corresponding participant whose demonstrations were used to train the agent. The control bar shows the performance of the control group participant who competed against the agent. High similarity between agent and experimental metrics, along with low similarity to control metrics, indicates strong imitation quality. Metrics where agent and control values are similar suggest weaker evidence of imitation.



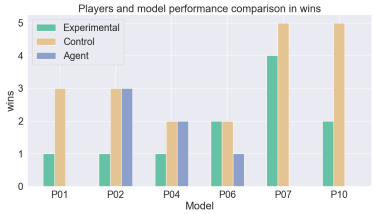
(a) Number of advantage attacks made by player or agent of measured team. There is no similarity present in regards of advantage attacks between player and agents.



(b) Reported disadvantage attacks by player or agent of measured team. Experimental participants and agents perform disadvantage attacks more frequently, suggesting possible similarity.



(c) Number of effective attacks made by player or agent in measured team. There is no similarity present in regards of effective attack behavior.



(d) Number of rounds won by each player, over five a total of five rounds. Participants in experimental group show lower win rate, as well do the agents.

Fig. 13: Comparison of agent and participant performance across in-game metrics. Similar to Figure 12, the agent bars represent Mirror Mode agents trained on demonstrations from experimental-group participants ($N = 6$, one model per participant). The experimental bars show the performance of the corresponding participant whose gameplay was used to train the agent. The control bars show the performance of the control group participants who competed against the agents. High similarity between agent and experimental metrics, combined with low similarity to control metrics, indicates strong imitation quality. Metrics where agent and control values are similar suggest weaker imitation.

trast, P04, and P06 use fewer movement actions, while P07 shows considerably more movements than its participant.

When comparing the death rate in Figure 12b from the Mirror agents to the participants, it is noticeable that the Mirror agents show a higher death rate, than most participants. However, in most models the death rate closely matches that of the experimental participant it was trained on. Suggesting that agent's skill level is adapted to that of the corresponding player. Notably, the death rates for P06 and P07, are higher than the rates from their participants. Similarly, the kill rates for P06, P07, and P10 are substantially lower than those of their corresponding participants.

It can be noted that the Mirror agents generally show low similarity to the control participants, supporting the idea that they are primarily imitate the strategies of the experimental group players. However, Figures 13c and 13d indicate lower similarity with the experimental participants for these metrics, highlighting less accurate imitation in offensive tactics.

These observations suggest that while Mirror agents effectively replicate movement behavior from the experimental players, they lack behind in imitating offensive strategies. Nonetheless, the alignment in death rates with the original players indicates a degree of skill-level adaption in the trained models.

Questionnaire The participants' replies to the questionnaires from the Standard Mode and Mirror Mode are compared to each other to find an influence on the player's game experience. Analysis of the two survey results revealed several insights. Overall, participants reported a shift in gameplay strategy, moving from defensive to more offensive tactics transitioning from Standard Mode to Mirror Mode.

Player's satisfaction was measured using a 1–5 rating scale across nine questions. Participants rated each question for both Standard Mode and Mirror Mode. For each question, the difference in ratings was calculated across all participants, separately for the experimental and control groups. The mean difference in score across all participants in both groups is computed and presented in Figure 14. Error bars represent the standard error of the mean (SEM), providing an indication of the precision of the estimated group means.

Enemy tactics were rated less challenging in Mirror Mode than in Standard Mode in both groups. Predictability and variety of enemy behavior was rated slightly less for Mirror Mode for the experimental group, but not for the control group. However, in Mirror Mode the adaption of the enemy tactics to the player's tactics scored much higher compared to Standard. The mean score difference increased by more than one point, with relatively small error bars indicating higher confidence that the group mean is close to the observed value. Therefore, there is a more noticeable adaptability of enemy strategies in the experimental group, where the enemy was actually adapted to their own tactics. The experimental group also felt more motivated to adjust their tactics in Mirror Mode.

Interestingly, the Control group found the enemy tactics less predictable and more varied compared to the experience of Experimental participants. Only minor differences were observed in player's motivation to continue playing, their sense of rewarding gameplay, and their perception of enemy's behavior being interesting.

Some questions show relatively large SEM values, indicating uncertainty in the mean score differences. In particular, player performance and rewarding gameplay show a high SEM value, suggesting these scores may not be accurately represent the underlying sample group. The larger error bars may partly reflect the small number of participants in each group.

Furthermore, participants' description of the enemy behavior differences of the two modes particularly also indicate the perception of the difference in of-

fensive and defensive play style of the enemies, as shown in Table 4. Participants in the experimental group noticed the mirroring behavior from the enemies in Mirror Mode, imitating their defensive behavior from the Standard game, such as P10 who is staying back, P06 who is less protective towards the magic unit, and P01 who is more likely to retreat. Only a few in the control group recognized their own strategies in the enemy tactics in Mirror Mode, such as P05 and P12, than in the control group.

From the provided scale rating questions, satisfaction scores are calculated as the mean of all ratings for each participant in both Standard Mode and Mirror Mode. The computed satisfaction scores for each participant are presented in Figure 15. Overall, most participants gave a higher satisfaction score to Mirror Mode than to Standard Mode, indicating a positive influence on their game experience by Mirror Mode. However, for each participant both satisfaction scores are high, and only small differences are noticed. The standard deviations are generally similar across the two modes, indicating comparable variability in responses overall and similar range of satisfaction. However, variability differs notably between participants. Participant P04 is particularly noteworthy, showing a very high standard deviation in Standard Mode but zero deviation in Mirror Mode, highlighting the increase in satisfaction for Mirror Mode.

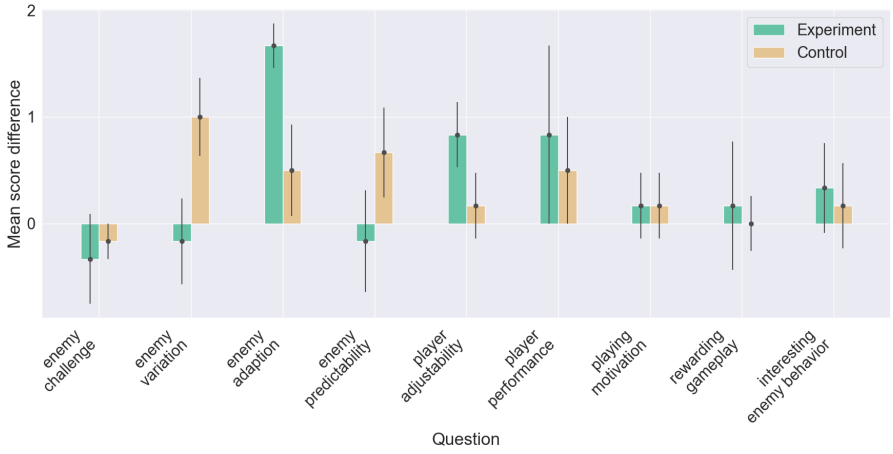


Fig. 14: Mean score differences per question type between Standard Mode and Mirror Mode for the experimental group (green) and control group (yellow). Error bars represent SEM. The y-axis lists question themes The x-axis shows the difference in mean score, with differences calculated as $MirrorMode - StandardMode$. Higher values indicate greater satisfaction in Mirror Mode, while smaller SEM reflects greater confidence. Results suggest increased satisfaction in enemy adaptability and player adjustability, but a decline in perceived challenge.

ID	Group	Response
P05	B	I noticed that the enemy behaviour changed with respect to my performance, and it was harder than before.
P03	B	I feel like the enemy backed off more often in the 2nd mode. And also that they attacked one of my people that I brought forward more often, so I could adjust my strategy better.
P02	A	The enemy behavior was more unpredictable which kept it very interesting
P08	B	It was smarter, less greedy and taking steps more carefully. It would sometimes retreat or do surprise attacks which was unexpected.
P09	B	Enemies would walk further away from you making you chase. Enemies did not always go on the offensive like it seemed in the first round.
P11	B	Honestly i thought they were way more defensive rather than just attacking every time they could:)
P10	A	The enemy was staying back more, as I did.
P06	A	Second mode used the same units as me. Also, the magic unit was not protected as well which is a habit I have.
P01	A	It mirrored my strategy. So it kept escaping rather than attacking.
P04	A	Played more safe and felt really dynamic given the game state
P07	A	The biggest difference was that the enemy decided to reposition a lot more and play more defensive, which made it a lot more challenging to attack effectively.
P12	B	They run away more. Used the strategy I used in the first 5 rounds but less smart executed

Table 4: Experimental (white) and control (grey) group participants replies to the question “What differences did you notice in the enemy behavior from the Second Mode compared to the First Mode?". First Mode corresponds to the Standard Mode, and Second Mode to the Mirror Mode. Participants especially noticed the change from offense to defense. P01, P06, P10 remarked the enemy’s behavior mirroring the player’s tactics.

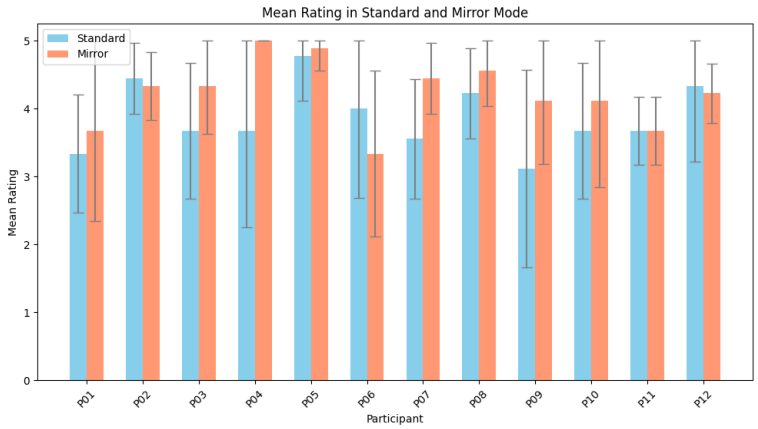


Fig. 15: Satisfaction scores for each participant in Standard Mode (blue) and Mirror Mode (red). Satisfaction score equals the mean rating across all questions in the survey, with error bars indicating standard deviation. Overall, satisfaction is slightly higher for Mirror Mode, with most participants showing similar standard deviations for both modes. Notably, P02 and P06 reported lower satisfaction in Mirror Mode. These results suggest that, for most participants, Mirror Mode provided a higher level of satisfaction compared to Standard Mode.

Total satisfaction scores for each mode and participant are provided in Table 6 in the Appendix. Comparisons of overall satisfaction between the two modes show no significant differences. A two-tailed paired t-test with $\alpha = 0.05$ resulted in $p = 0.2566$ for the experimental group and $p = 0.1144$ for the control group. Moreover, there was also no significant difference found in score difference between the two groups, as the statistical test gave a value of $p = 0.9153$. A summary of the computed t-test results are presented in Table 5.

It is noticeable that P06 rates the Mirror Mode lower than other participants in the experimental group, making P06 an outlier. Computing the statistical t-test without P06 still shows no significant difference among the experimental group, resulting in $p = 0.0794$.

Therefore, the overall game satisfaction of players did not seem to differ significantly.

Comparison	**p-value**	**Significance**
Experimental group: Standard vs Mirror Mode	0.2566	Not significant
Control group: Standard vs Mirror Mode	0.1144	Not significant
Experimental vs Control groups	0.9153	Not significant

Table 5: Results of statistical paired t-test with $\alpha = 0.05$, of satisfaction scores between modes and groups. No significant differences were observed.

7 Discussion

Statistics have shown a reduction in the popularity of strategy video games [34], potentially being caused by the repetitive and predictable behavior from NPC enemies [5][18]. Prior studies have found advanced techniques to incorporate AI with video games, making it more enjoyable for players [2]. Building on this, this study introduced a new game mode in strategy video games called Mirror Mode, where the enemy agents are trained on the playing behavior of players so the agents mimic their strategy.

7.1 Reflection on Experimental Results

The experimental results demonstrate promising indications of imitation in Mirror Mode, particularly in the replication of defensive behaviors such as unit retreating and hesitant attacks. Despite the shown effectiveness of the trained models, further improvement is necessary for the agents to fully mimic a player's overall strategic approach. Offensive tactics proved more challenging to replicate, with agents frequently holding back to initiate attacks or failing to exploit advantageous or effective actions. A likely explanation lies in the greater complexity of the decision space for executing valid attacks compared to performing simple movement actions. Successful attacks require the agent to first identify an appropriate target within range, then position itself on a tile from which the attack can be performed. This involves at least two additional decision steps, whereas a basic movement requires only the selection of a single tile within the agent's movement range. This difference makes the probability of selecting a valid movement action higher than that of selecting a valid attack, which may account for the observed performance gap.

The lack in offensive replication could also partially be caused by the difficulty of the standard enemy AI. While both the player as well as the agents struggled to defeat the standard AI, it is difficult to identify if difficulty had a beneficial impact on the results. During Standard Mode, players showed a higher death rate, and rated the enemy AI as more challenging. The Mirror Mode agents displayed similar struggles when facing standard AI opponents. This suggests a potential domino effect, where players who performed suboptimally in demonstrations, inadvertently transferred those struggles to their agents. Consequently, the agents inherited and reproduced similar results during training and their final model.

Despite the model performance, participants rated Mirror Mode satisfaction generally higher than Standard Mode. However, the difference in satisfaction scores between the two modes was not statistically significant within either the experimental or control group. Likewise, no significant differences were observed between the groups themselves. Two factors have had a possible influence on the satisfaction scores, and the differences between the two modes. In some cases, lower satisfaction with Mirror Mode appeared to be caused by the less challenging enemy behavior compared to Standard Mode. Furthermore, all participants were already familiar with strategy games, and most reportedly enjoying the

standard games. As a result, baseline satisfaction with Standard Mode was already high, leaving limited room for improvement in measured enjoyment. In addition, Mirror Mode frequently repeated defensive actions, which made its behavior more predictable over time and consequently reduced both challenge and engagement.

7.2 Limitations

In this study, the agents for Mirror Mode were trained for a total of 1 million steps due to the limited time frame available. This training duration may have been insufficient for agents to effectively learn and adapt offensive strategies, leading to a limited execution of valid offensive behaviors and a less realistic replication of a player's strategies. The training process progressed at a rate of approximately one hour per 100,000 steps, resulting in considerable computational demands. With a longer time frame, however, it would be feasible to extend training beyond 24 hours, potentially enabling more comprehensive learning and improved imitation behavior.

Additionally, the model configuration used for training Mirror Mode agents requires refinements. While the GAIL loss metric provides a general indication of imitation performance, it does not directly measure the quality of imitation within actual gameplay. This creates a gap between training evaluation and in-game behavior, limiting the model imitation expectations through model fine-tuning evaluations.

Moreover, the reward system needs to be explored better to improve the imitation quality. The current reward system rewards agents for winning the game and defeating enemies, while penalizing them for losing or being killed. This design primarily emphasizes overall game performance and survival rather than to closely imitate player demonstrations. Consequently, achieving an appropriate balance between exploration and imitation was necessary during model configuration. A potentially more effective approach could involve rewarding agents for executing actions that closely match those in the original player demonstrations. This would require the option to retrieve exact state-action pairs from the recorded demonstrations, which still needs to be explored within the Unity ML-Agents toolkit. Once possible, this could provide an interesting method for mimicking agent behavior.

Finally, certain limitations in the game mechanics may have unintentionally increased the game difficulty. In the original version of the strategy game,units have access to special skills, abilities, and assists that enhance their statistics and overall power, making victories easier to achieve while also requiring more strategical thinking.

7.3 Future Work

Future research should address the aforementioned limitations to improve both the effectiveness and engagement of Mirror Mode. Exploring alternative configurations for training the Mirror Mode agents could lead to better imitating

behavior of offensive behavior and overall strategies. Parameters such as hidden units and batch size, can be further tested and finetuned, to find a more optimal model.

Moreover, in future research, a more detailed in-game evaluation can be used to assess imitation quality from a player's perspective. Generating multiple candidate models based on promising performance metrics and testing them in actual gameplay could provide a clearer understanding of how training outcomes translate into in-game behavior. This evaluation should focus on strategic imitation, similar to the approach used in this study, but expanded to include more detailed player assessments of imitation quality.

Refinements to the reward system are also necessary. In particular, providing explicit rewards for replication actions from recorded demonstrations. This could be a beneficial approach to a stronger imitation performance. Ultimately, finding a more suitable Mirror Mode model and reward system could produce agents that are not only more effective but also more engaging in gameplay, possibly increasing participant's satisfaction.

Furthermore, future work could consider incorporating more advanced game elements, such as special skills, abilities, and assists. This could help balance gameplay difficulty, and potentially increase player's satisfaction scores.

Finally, an ideal implementation of Mirror Mode would last over an extended period of player gameplays, allowing the model to continuously integrate new demonstrations and adjusts to new player strategies that involve counter strategies developed against their own behavior. This long-term adaption would repeatedly adjust strategies, encouraging players to think more strategically while reducing predictable gameplay patterns and repetitive actions. To support this, future user experiments should be conducted over longer durations, using multiple models that are progressively updated across several gameplay sessions. In order to achieve this, it would be beneficial tointegrate Mirror Mode in a mobile version of the game, enabling participants to play on their own devices. This would require a faster and more flexible training service, potentially supported by cloud-based services, to provide the necessary connections and computational resources for mobile compatibility.

8 Conclusion

This research introduces a new gameplay mode called Mirror Mode, for turn-based strategy games. It is found that RL and IL techniques from the Unity ML-Agent package provided by Juliani et al. [16] show big potentials for teaching agents to copy player strategy behavior, as after only 200k steps the agents showed learning progression. However, an optimal configuration needs to be applied for the best imitation behavior, as it is difficult to maintain a balance in imitation and exploration.

Player tests were conducted to evaluate the Mirror Mode model configuration found in the first set of experiments. The tests provided great insights in regards of player satisfaction and agent imitation abilities. A significant paired t-test

showed no significant change in satisfaction score when going from Standard Mode to Mirror Mode for the experimental group as well as the control group in the player tests. Moreover, there was no significant difference between the two groups. Therefore, the null hypothesis that the introduction of Mirror Mode has no impact on the player's satisfaction, cannot be rejected. However, for most participants the satisfaction score was higher in Mirror Mode compared to Standard Mode. The game challenge was rated easier for Mirror Mode, but overall players enjoyed it better because of the less predictable behavior.

Additional metrics taken from the participant's gameplay indicate a good imitation in defensive behavior rather than offensive tactics. Participants recognized their own retreating tactics, and laid back approach, in the Mirror Mode enemies. Recorded game metrics compare the participant playing behavior to the agent behavior from their Mirrored model. These results confirm the imitation capability of the defensive behavior from the participants.

In addressing the question "*How will a player's game experience be influenced when NPCs imitate their strategy in a turn-based strategy game?*", this study therefore shows that no significant difference in satisfaction between Mirror Mode and Standard Mode can be found. Concluding that no confirmation can be done about any significant influence on game experience. However, overall participants enjoyed Mirror Mode better due to the less predictable enemy behavior and their recognized defensive strategies, but making them easier to defeat.

The first half of the study focused on providing a model for Mirror Mode. Leading to an answer to the question: "*To what extent can RL and IL be applied to teach NPCs a player's strategy in a turn-based strategy game?*" It can be concluded that IL has strong potentials in teaching enemy agents a player's strategy, with the proper configurations to maintain the imitation-exploration trade-off. RL is more adaptable in finding its own strategy, and less suitable for copying player's strategies. Therefore, IL is more preferable for the specific purpose of teaching NPC's a player's strategy, with PPO and extrinsic rewards added for a good agent performance. Lastly the hypothesis *The game experience of a player will be positively influenced after the enemy AI has imitated their strategy, as the player will be more engaged and more satisfied with the game.* can thus not be confirmed. However, survey results indicate an impact on experience in regards of noticeably different enemy behavior adapted to the player's performance, and an overall increase in satisfaction.

Whether Mirror Mode can increase the popularity in strategy games remains to be discussed, but this study takes the first strategic step toward that direction.

Acknowledgement

First of all, this project wouldn't have been possible without Code Monkey's YouTube tutorials [7], which were a huge help along the way. Also, a minor thanks to the Fandom Community and Fire Emblem Heroes Wiki for providing a public space for UI and character sprites of the Fire Emblem games [12].

Additionally, I want to thank my supervisors, Professor P. van der Putten and Professor A. Plaat, for their guidance and advice, which really helped me stay on track and focus on my goals.

And of course, I'm very grateful to my 12 participants for taking part in my research and for their positive and enthusiastic attitude. Their participation really helped me to achieve my goal.

References

1. M. Alonso Jr.: ML-Agents Installation (2024), https://github.com/Unity-Technologies/ml-agents/blob/release_22_docs/docs/Installation.md, (most recently accessed on June 2025)
2. Akram, A., Tehseen, R., Saqib, S., Nazir, F., Awan, M., Jr, I.: Advanced AI Mechanics in Unity 3D for Immersive Gameplay. A Study on Finite State Machines & Artificial Intelligence. International Journal of Innovations in Science and Technology **6**, 2004–2023 (12 2024)
3. Amato, C., Shani, G.: High-level reinforcement learning in strategy games (01 2010). https://doi.org/10.1145/1838206.1838217
4. Baker, B., Kanitscheider, I., Markov, T., Wu, Y., Powell, G., McGrew, B., Mordatch, I.: Emergent Tool Use From Multi-Agent Autocurricula (2020), https://arxiv.org/abs/1909.07528
5. Chanel, G., Rebetez, C., Bétrancourt, M., Pun, T.: Boredom, engagement and anxiety as indicators for adaptation to difficulty in games. In: Proceedings of the 12th International Conference on Entertainment and Media in the Ubiquitous Era. p. 13–17. MindTrek '08, Association for Computing Machinery, New York, NY, USA (2008). https://doi.org/10.1145/1457199.1457203, https://doi.org/10.1145/1457199.1457203
6. Chang, SC., C.Y..H.J.: Determining Satisfaction from Gameplay by Discussing Flow States Related to Relaxation and Excitement (03 2020). https://doi.org/https://doi.org/10.1007/s40869-020-00113-5
7. Code Monkey: How to use Machine Learning AI in Unity! (ML-Agents) (2020), https://unitycodemonkey.com/video.php?v=zPFU30tbyKs
8. Csikszentmihalyi, M.: Beyond Boredom and Anxiety. Jossey-Bass Publishers (1975), https://books.google.nl/books?id=afdGAAAAMAAJ
9. Fan, X.: The application of reinforcement learning in video games (2023). https://doi.org/10.2991/978-94-6463-300-9_21, https://doi.org/10.2991/978-94-6463-300-9_21
10. Fandom: List of heroes (2025), https://feheroes.fandom.com/wiki/List_of_Heroes
11. Fire Emblem Heroes (FEH) Walkthrough Team: How Does Enemy AI Work? (2021), https://game8.co/games/fire-emblem-heroes/archives/324532
12. Fire Emblem Heroes Wiki: Game assets collection (2025), https://feheroes.fandom.com/wiki/Game_assets_collection#UI_Sprite_sheets

13. Gharbi, H., Fennan, A., Lotfi, E.: REPLICATING VIDEO GAME PLAYERS' BEHAVIOR THROUGH DEEP REINFORCEMENT LEARNING ALGORITHMS. Journal of Theoretical and Applied Information Technology **102**, 5735 (08 2024)
14. Ho, J., Ermon, S.: Generative Adversarial Imitation Learning (06 2016). https://doi.org/10.48550/arXiv.1606.03476
15. Huang, R.: The Impact of Flow State and Immersion in Video Games. Communications in Humanities Research **5**, 43–48 (09 2023). https://doi.org/10.54254/2753-7064/5/20230028
16. Juliani, A., Berges, V.P., Teng, E., Cohen, A., Harper, J., Elion, C., Goy, C., Gao, Y., Henry, H., Mattar, M., Lange, D.: Unity: A general platform for intelligent agents. arXiv preprint arXiv:1809.02627 (2020), https://arxiv.org/pdf/1809.02627.pdf
17. Lee, M.S., Heeter, C.: What do you mean by believable characters?: The effect of character rating and hostility on the perception of character believability. Journal of Gaming & Virtual Worlds **4**(1), 81–97 (2012). https://doi.org/https://doi.org/10.1386/jgvw.4.1.81_1, https://intellectdiscover.com/content/journals/10.1386/jgvw.4.1.81_1
18. Lemaitre, J., Lourdeaux, D., Chopinaud, C.: Towards a Resource-based Model of Strategy to Help Designing Opponent AI in RTS Games (01 2015). https://doi.org/10.5220/0005254402100215
19. Manaloto, N.: How naughty dog made enemies smarter in the last of us 2 (2020), https://www.ungeek.ph/2020/06/how-naughty-dog-made-enemies-smarter-in-the-last-of-us-2/
20. Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., Riedmiller, M.: Playing Atari with Deep Reinforcement Learning (2013), https://arxiv.org/abs/1312.5602
21. NotMike95: Chapter 6: Part 3 "Blunt Princess" | Fire Emblem Heroes [31] | NotMike95 (2017), https://www.youtube.com/watch?v=sPyXdv7vDXs, youTube video
22. OpenAI, Berner, C., Brockman, G., Chan, B., Cheung, V., Dębiak, P., Dennison, C., Farhi, D., Fischer, Q., Hashme, S., Hesse, C., Józefowicz, R., Gray, S., Olsson, C., Pachocki, J., Petrov, M., de Oliveira Pinto, H.P., Raiman, J., Salimans, T., Schlatter, J., Schneider, J., Sidor, S., Sutskever, I., Tang, J., Wolski, F., Zhang, S.: Dota 2 with Large Scale Deep Reinforcement Learning (2019), https://arxiv.org/abs/1912.06680
23. Pagulayan, R., Keeker, K., Wixon, D., Romero, R., Fuller, T.: User-Centered Design in Games. Human-Computer Interact. Handb pp. 883–906 (01 2003). https://doi.org/10.1201/b11963-39
24. Ross, S., Bagnell, D.: Efficient Reductions for Imitation Learning (13–15 May 2010), https://proceedings.mlr.press/v9/ross10a.html
25. Ross, S., Gordon, G.J., Bagnell, J.A.: A Reduction of Imitation Learning and Structured Prediction to No-Regret Online Learning (2011), https://arxiv.org/abs/1011.0686
26. Schulman, J., Wolski, F., Dhariwal, P., Radford, A., Klimov, O.: Proximal Policy Optimization Algorithms (2017), https://arxiv.org/abs/1707.06347
27. Smid, Y.: Masterthesis (2025), https://github.com/YannaSmid/MasterThesis
28. Sánchez-Ruiz, A., Stephen, R., Héctor, L.U., Noz-Avila, M., Díaz Agudo, B.: Game AI for a Turn-based Strategy Game with Plan Adaptation and Ontology-based retrieval (07 2008)
29. The Game Awards: Winners (12 2020), https://thegameawards.com/rewind/year-2020

30. Unity Technologies: Reward signals (2020), https://github.com/Unity-Technologies/ml-agents/blob/0.15.0/docs/Reward-Signals.md
31. Unity Technologies: Unity ML-Agents Toolkit (2022), https://unity-technologies.github.io/ml-agents/Training-Configuration-File/#gail-intrinsic-reward
32. Unity Technologies: ML-Agents Overview (2024), https://docs.unity3d.com/Packages/com.unity.ml-agents@3.0/manual/index.html
33. Weng, L.: From GAN to WGAN (2019), arXiv:1904.08994
34. Yee, N.: Gamers have become less interested in strategic thinking and planning (05 2024), https://quanticfoundry.com/2024/05/21/strategy-decline/
35. Zare, M., Kebria, P.M., Khosravi, A., Nahavandi, S.: A Survey of Imitation Learning: Algorithms, Recent Developments, and Challenges (2023), https://arxiv.org/abs/2309.02473

A Appendix

A.1 Game Environment Setup

In Unity, a new 2D game environment needs to be created. The game environment was built through the following steps:

1. Setup a simple 2D game environment in **Unity project editor version 2023.2.13f1**.
2. **The ML-Agent toolkit v3.0.0** package is installed in the Unity editor [32] developed by A. Juliani et al. [16]. The backend implementation for the ML algorithms are implemented by A. Juliani et al., and are accessible through the ML-Agent package.

A.2 Virtual Environment

Creating a virtual environment in the Unity project directory can either be done in via Conda, or the local Command Prompt on Windows. This study used Command Prompt.

1. Navigate to the Unity project directory in Command Prompt.
2. Create a venv folder by running the following command **python** *-m venv venv.*
3. Activate the environment with: *venv \ scripts \ activate.*
4. Ensure that **Python version 3.10.11** is installed and active within the virtual environment.
5. Upgrade pip through **python** *-m pip install –upgrade pip.*
6. Install the ML-Agents toolkit with **pip** *install mlagents.*
7. Install PyTorch version 2.2.1 using **pip3** *install torch==2.2.1.*

The **mlagents-learn** commands can now be used effectively to communicate with the Unity environment. To check if the ML-Agents toolkit is installed correctly, the following command can be ran: **mlagents-learn** *–help.*

A.3 Configuration File

```
behaviors:
  MirrorMode:
    trainer_type: ppo
    hyperparameters:
      batch_size: 256
      buffer_size: 2048
      learning_rate: 0.0003
      beta: 0.001
      epsilon: 0.2
      lambd: 0.95
      num_epoch: 3
      learning_rate_schedule: linear
    network_settings:
      normalize: false
      hidden_units: 256
      num_layers: 2
      vis_encode_type: simple
    reward_signals:
      extrinsic:
        strength: 0.5
        gamma: 0.99
      gail:
        gamma: 0.9
        strength: 1.0
        network_settings:
          normalize: true
          hidden_units: 128
          num_layers: 2
          vis_encode_type: simple
          memory: null
          goal_conditioning_type: hyper
          deterministic: false
        learning_rate: 0.0001
        encoding_size: null
        use_actions: true
        use_vail: false
        demo_path: [...]
    init_path: null
    keep_checkpoints: 5
    even_checkpoints: false
    max_steps: 1000000
    time_horizon: 64
    summary_freq: 2000
    threaded: false
```

```
    self_play: null
    behavioral_cloning:
      demo_path: [...]
      steps: 900000
      strength: 0.4
      samples_per_update: 10
      num_epoch: null
      batch_size: 512
```

A.4 Participant Experience and Skills Information

participantID

participantID	How would you rate your experience with strategy video games? (1 = No experience, 5 = Very experienced)	How enjoyable do you find strategy video games? (1 = Not enjoyable, 5 = Very enjoyable)
P01	2	3
P02	5	5
P03	3	4
P04	5	5
P05	3	4
P06	5	5
P07	4	4
P08	5	4
P09	5	5
P10	4	5
P11	4	4
P12	5	5

Fig. 16: Experience and skill level of each participant, rated by the participants themselves.

A.5 Total Satisfaction Scores Standard Mode and Mirror Mode

ID	Group	Standard Mode	Mirror Mode	Score Difference
P01	A	30	33	3
P02	A	40	39	-1
P03	B	33	39	6
P04	A	33	45	12
P05	B	43	44	1
P06	A	36	30	-6
P07	A	32	40	8
P08	B	38	41	3
P09	B	28	37	9
P10	A	33	37	4
P11	B	33	33	0
P12	B	39	38	-1

Table 6: Total satisfaction scores for each participant per mode. Satisfaction scores slightly higher for Mirror Mode. P02 and P06 reported lower satisfaction in Mirror Mode. The results suggest that, for most participants, Mirror Mode gave a higher level of satisfaction compared to Standard Mode.