Annales Mathematicae et Informaticae
61 (2025) pp. 80–93
DOI: 10.33039/ami.2025.10.017
URL: https://ami.uni-eszterhazy.hu

Making the boss smarter: A journey from rule-based to learned behaviors

Réka Erdész, Ede Troll

Eszterházy Károly Catholic University
reka.erdesz@uni-eszterhazy.hu
troll.ede@uni-eszterhazy.hu

Abstract. This paper investigates the application of reinforcement learning to non-player character (NPC) behavior design in a roguelike video game, with the goal of creating more engaging and less predictable opponents. Two training approaches were compared using the Unity ML-Agents framework: Agent A, trained exclusively through self-play, and Agent B, trained initially against human players before switching to self-play. Performance was evaluated using quantitative metrics such as policy loss, value loss, entropy, and ELO ratings, alongside qualitative feedback from semi-professional players. While Agent B achieved faster convergence and higher ELO scores, player feedback indicated a preference for Agent A due to its unpredictability, balanced tactics, and lower frustration levels. The results highlight the need to balance technical optimization with player experience, and suggest that hybrid training strategies may yield the most compelling adversaries in future game AI design.

Keywords: reinforcement learning, game AI, Unity ML-Agents, player experience, NPC behavior

AMS Subject Classification: 68T05, 68T07, 91A35

1. Introduction

This study examines the integration of reinforcement learning techniques into the design of non-player character (NPC) behaviors within a roguelike video game environment. The research aims to address limitations inherent in traditional rule-based artificial intelligence (AI) approaches, which often lead to predictable and

Accepted: October 15, 2025
Published online: October 28, 2025

monotonous adversary behavior, thereby reducing gameplay challenge and engagement [11]. Notable successes in reinforcement learning, such as mastering the game of Go through self-play [9], highlight the potential of these techniques to create adaptive and challenging opponents.

The initial implementation employed a rule-based AI [2], utilizing the A* pathfinding algorithm for navigation and a behavior tree [5] for the final boss. Although the behavior tree introduced conditional actions – such as retreating when the agent's health was low or becoming aggressive when the player was weakened – the decision-making process remained deterministic and easily exploitable by experienced players.

To overcome these constraints, We implemented two alternative AI training methodologies using the Unity Machine Learning Agents (ML-Agents) framework [4]. Both agents were trained via reinforcement learning [10], enabling them to improve their decision-making through interaction with the game environment. Agent A was trained entirely through self-play, iteratively competing against its previous policy versions. Agent B underwent an initial training phase of 30,000 steps against human players before switching to self-play. This dual approach allowed us to compare the benefits of purely autonomous learning against a hybrid method incorporating human-guided exploration.

The primary objective of this research is to evaluate which training paradigm yields a more challenging, engaging, and strategically capable opponent. Evaluation was conducted through both quantitative performance metrics and qualitative human player feedback, with the ultimate goal of identifying design practices that enhance player experience while maintaining balanced gameplay difficulty.

1.1. Game concept and the original enemy AI

The game [7] is a room-based rogue-like video game. In each room, the player has to defeat randomly generated enemies, fighting their way to the final boss. Initially, enemies were operating on a rule-based system; their movements were based on the A* algorithm. While the smaller enemies only had a few attack types, the final boss had a so-called *Behavior Tree* [5], which resulted in a better experience. The behavior tree gave the boss some human-like behavior based on the health percentage of its own and its opponent's – the player's health. It retreated when it had a low health percentage, and it became aggressive when the player had low health – but it was still predictable. The agent's actions were defined by specific rules, making the enemy's responses monotonous and easy to anticipate. This predictability reduced the overall gameplay challenge.

1.2. Teaching process of the enemy

For the research [3], We used the Unity ML-Agents toolkit for teaching purposes. This is an open-source toolkit developed by Unity Technologies in 2020. The core concept involves an Agent that learns through reinforcement learning. Unity provides a plug-in package that includes the necessary code libraries and connects to

the virtual environment. To get started, We had to create a virtual environment and, like with all machine learning projects, We needed to install TensorBoard and PyTorch using pip. Additionally, We installed ML-Agents, which requires NumPy and TensorFlow.

1.2.1. ML-Agents

The ML-Agents framework is centered around reinforcement learning, where an agent learns to make decisions in various situations based on feedback from its environment, known as rewards. This learning process does not rely on examples from teachers; instead, it is driven by self-trial and error. The feedback received from the environment aids in developing an optimal strategy for the agent.

Formally, this process is described by a Markov Decision Process (MDP) [6], whose components are:

$$\mathcal{M} = \langle S, A, P, R, \gamma \rangle$$

An MDP is a mathematical framework used to model decision-making in environments where outcomes are partly random and partly under the control of the agent.

State space (S) The state space defines all observable features available to the agent during decision-making. In this study, the agent's state vector consisted of positional, health, status, timing, and event-related variables, as summarized in Table 1.

Table 1. Summary of state space components.

Category	**Description**
Positional data	Distance from the player; direction of the player relative to the agent; agent's facing direction.
Health information	Current health points of the agent; current health points of the player.
Status flags	Attack availability (boolean flag preventing overlapping attacks); whether the agent is currently attacking.
Timing	Time elapsed since the last attack.
Special events	Whether the player is affected by the "black hole" status (immobilized).

The first experiment We conducted was a test on reinforcement learning with a hardcoded AI opponent. We trained an Agent against this opponent, and this is where We first initialized the reward system. Initially, the rewards were set too high, which made the agent's policy unstable. To address this, We normalized the large integers to a $0 \dots x$ interval, resulting in more stable learning.

Although it is important from a teaching perspective to know how much reward the agent received and why, the essence of the research lies in the methodology, which can be particularly relevant in roguelike games. The reward system will differ for each developer's model; as an extension, we aim to apply this methodology to other genres and multiple games.

Action space (A) Table 2 lists the discrete actions available to the agent at each decision step.

Table 2. Summary of action space components.

Action	**Description**
Move forward/backward	Adjusts the agent's position along the facing direction.
Strafe left/right	Moves perpendicular to the current facing direction.
Rotate left/right	Changes the facing direction of the agent.
Basic attack	Executes the standard melee or ranged attack, depending on the agent's class.
Special abilities	Uses special skills such as "black hole" or other area-of-effect attacks.
Retreat/Heal	Moves away from the player and recovers health if possible.
Idle	Performs no action for a single decision step.

Reward system (R) The agent received rewards for winning, using varied attacks (to discourage attack spamming), surviving with low health (encouraging prolonged combat), and performing tactical retreats when low on HP. We initially gave a reward for movement, but zig-zag movement developed, so we removed that reward. This led to the agent learning to move strategically on its own. General penalties were applied for taking damage, standing still, and missing hits.

Table 3 summarizes the conditions for rewards and penalties.

One of the key lessons learned during training was that relying solely on the reward system was insufficient. In many cases, manual adjustments to the weights of rewards and penalties were necessary to improve learning stability or to address unwanted behaviors.

The main part of the agent's reward system was the ability system. First, a base reward was introduced that the agent received for every reward event, based on the damage ratio:

$$D = \frac{damage}{playerCurrentHP}$$

Table 3. Reward and penalty conditions.

Event	**Type**
Winning a match	Reward
Successful attack hit	Reward
Varied attack usage	Reward
Survival with low health	Reward
Tactical retreat when low HP	Reward
Taking damage	Penalty
Standing still (no action)	Penalty
Missed attack	Penalty
Reward for movement (removed due to zig-zag behavior)	Removed reward

1.2.2. PPO

PPO – Proximal Policy Optimization [8], is a popular reinforcement learning algorithm often used in game development because it stably improves AI decisions without sudden big jumps in learning. This is especially important when the opponent is not deterministic, such as a human player, so the AI learns and adapts gradually. PPO constrains policy changes so that the ratio of new to old policies can only move within a certain range, thus facilitating the learning of complex strategies and the retention of prior knowledge.

We chose it for the project because it handles complex decision situations such as healing or tactical switching during combat well, and because it is natively supported by Unity ML-Agents, it is easy to integrate.

1.2.3. Agent A

Agent A was trained exclusively through self-play, meaning it fought against its previous versions, a technique inspired by successful applications like AlphaGo [9], which leveraged self-play to master complex strategic games. In the YAML configuration, we gradually increased the batch size (512), buffer size (5120), and entropy value (0.02), and linearly decreased the learning rate to ensure stable and continuous improvement. The design of the reward system was guided by principles of player-centric AI design [11], aiming to encourage varied and engaging behaviors rather than repetitive actions.

The self-play settings $includedsave\_steps = 20000$, $team\_change = 150000$, $swap\_steps = 5000$, which ensured that opponents were periodically refreshed. This prevented the agent from getting stuck in repetitive patterns and allowed it to learn against a variety of opponents. The selected model was chosen from the run that produced the lowest policy and value loss values.

1.2.4. Agent B

Agent B took its first 30 000 steps fighting against human players, then switched to self-play learning. With player guidance, the AI learned relevant patterns early on, its behavior stabilized more quickly, and it mastered tactical elements better – such as when to heal or avoid combat. With Agent B, we encountered the well-known problem in reinforcement learning: reward hacking. I mentioned this phenomenon in 1.2.1 that the agent found a loophole in the reward system with movement rewards. Our next research is based on this experience.

Hyperparameters

For reproducibility, we detail the complete PPO setup for both agent in Table 4.

Table 4. PPO hyperparameters used in training.

Parameter	Value
Learning rate	3×10^{-4}
Batch size	512
Buffer size	5120
Gamma (γ)	0.99
GAE λ	0.95
Entropy coefficient (beta)	0.001
Clip range (epsilon)	0.3
Number of epochs	5
Number of environments	1
Hidden units	128
Number of layers	2
Time horizon	64
Memory sequence length	64
Memory size	128
Seed	-1 (random)

Evaluation Protocol (Appendix)

The evaluation protocol was based on the self-play configuration in the YAML files. The details are as follows:

- **Opponent pool:** maintained as a sliding window of the last 10 models (defined by `window: 10`).
- **Checkpoint saving:** new models were added to the pool every 20,000 steps (`save_steps: 20000`).

- **Team change:** enforced after 150,000 steps (`team_change: 150000`).
- **Opponent swap frequency:** opponents were rotated every 5,000 steps (`swap_steps: 5000`).
- **Latest model ratio:** 50% of matches were played against the most recent model (`play_against_latest_model_ratio: 0.5`).
- **Initial ELO:** all agents started at 1200 (`initial_elo: 1200.0`).

1.3. Measurements

To evaluate the agents with metrics, we used TensorBoard and evaluated the following metrics to assess AI performance:

- **Policy Loss**: how stable the AI strategy is.
- **Value Loss**: how well it can predict future rewards, see Figure 1, 2.
- **Entropy**: the variability of decisions can be seen in Figure 3.
- **ELO**: the relative strength between different AI models, see Figure 4.

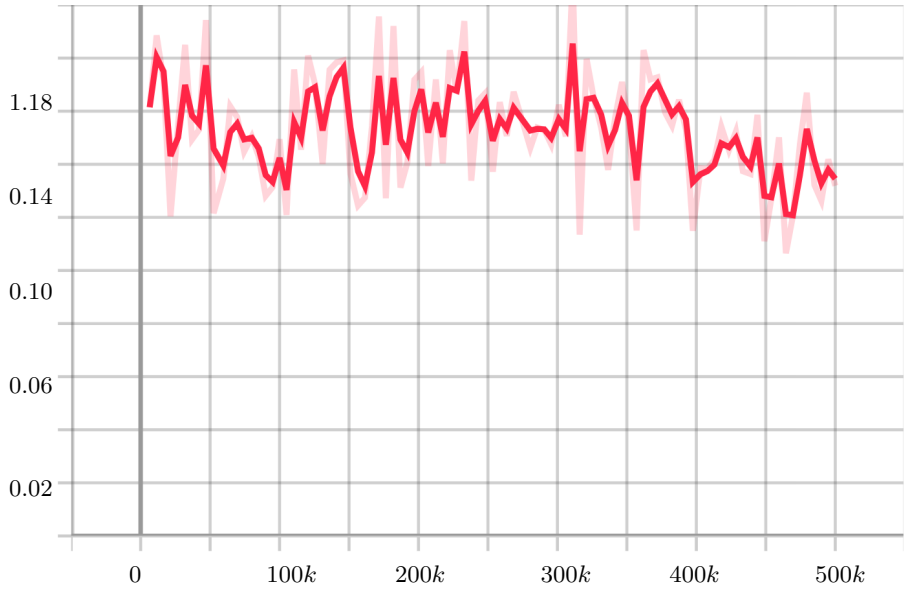

Figure 1. *Agent A*'s TensorBoard scalar.

The agent with initial intelligence, trained against the player, showed more stable and rapid progress. Its Policy Loss quickly converged to around 0.04, see

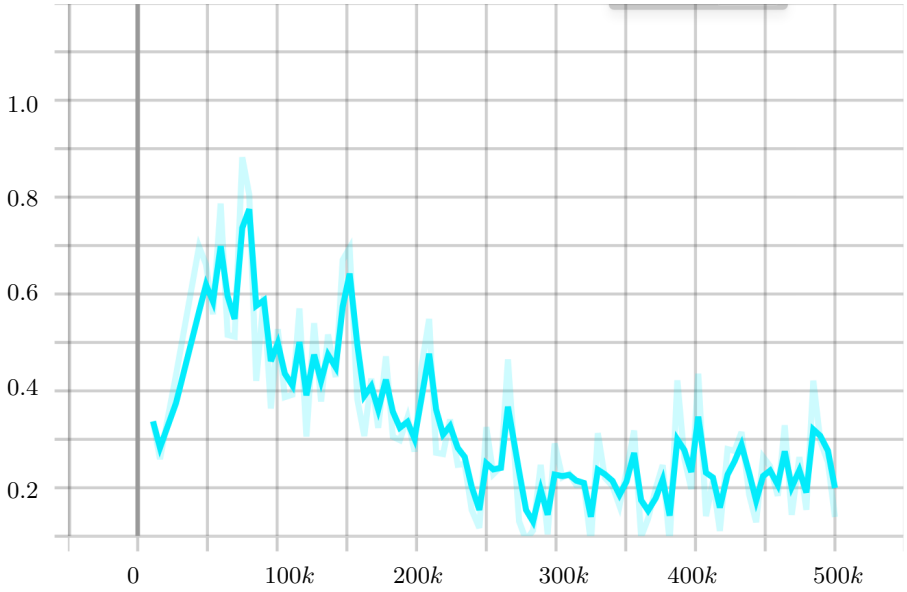

Figure 2. *Agent B*'s TensorBoard scalar.

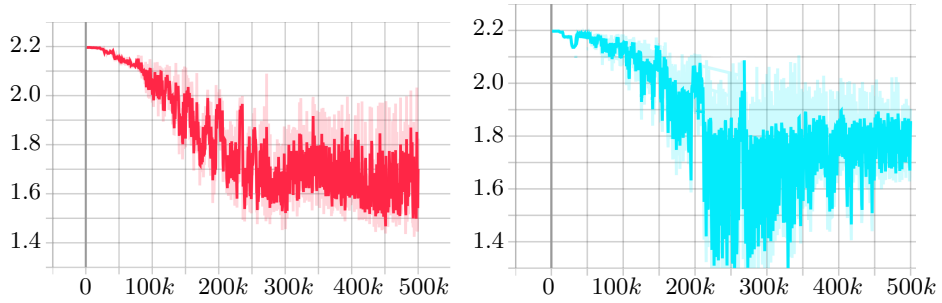

Figure 3. On the left diagram, *Agent A*'s TensorBoard scalar of entropy can be seen, while *Agent B* is on the right.

Figure 2, while the self-learning agent began to improve only after several hundred thousand steps, see Figure 1, exhibiting significant fluctuations. Both agents demonstrated a gradual decrease in Value Loss, indicating effective reward prediction.

Entropy values stabilized between 1.7, and 1.8, see Figure 3 for both agents, reflecting their ability to explore without becoming entirely random. However, the ELO curves highlighted differences: the agent with initial intelligence started with a higher ELO and maintained competitiveness, while the self-learning agent's ELO gradually declined, likely due to overly narrow strategies.

In summary, the agent that began with initial knowledge appears to be a more formidable enemy, based on the indicators above.

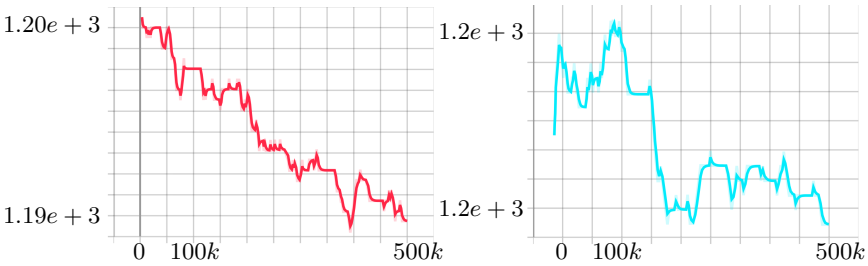

Figure 4. On the left diagram, *Agent A*'s TensorBoard scalar can be seen, while *Agent B* is on the right.

1.4. Results

To evaluate the behavior of the two AIs, we asked ten semi-professional players to try out the two versions. Participants completed a 22-question questionnaire that assessed gameplay experience, frustration, strategic depth, and predictability.

1.4.1. Gaming experience and preference

Agent A received an average rating of 4 out of 5, compared to Agent B's 2.6 out of 5 (Q4-Q5). Additionally, frustration levels were significantly different: Agent A scored 2.4 out of 5, while Agent B scored 4 out of 5 (Q15-Q16), suggesting more negative experiences with Agent B, see Figure 5.

1.4.2. Evaluation of strategy and tactics

It is interesting to note that, despite Agent A being more popular, 70% of players (7 out of 10) believed that Agent B had more advanced strategic skills (Q11). However, Agent A was perceived as more unpredictable (Q8), while Agent B's behavior was often predictable. In some cases, this predictability led to a negative gaming experience, particularly due to the overuse of the black hole skill, which received criticism from several respondents (Q13), see Figure 6.

1.4.3. Realism of AI Behavior

Participants assessed the realism of each agent's behavior to determine how closely they resembled a human opponent. According to Q18, the majority (6/10) perceived neither agent as particularly human-like, with 2/10 favoring Agent A and 2/10 favoring Agent B. Quantitative ratings further revealed that Agent B was perceived as slightly more realistic, with an average score of 3.1/5 (Q20) compared to Agent A's 2.6/5 (Q19). This difference likely stems from Agent B's more advanced strategic patterns, as evidenced by 90% of participants (Q11) noting its superior strategies. However, the relatively low realism scores for both agents suggest that further refinements are needed to enhance the perception of human-like decision-making, a critical factor for immersive gameplay in roguelike games.

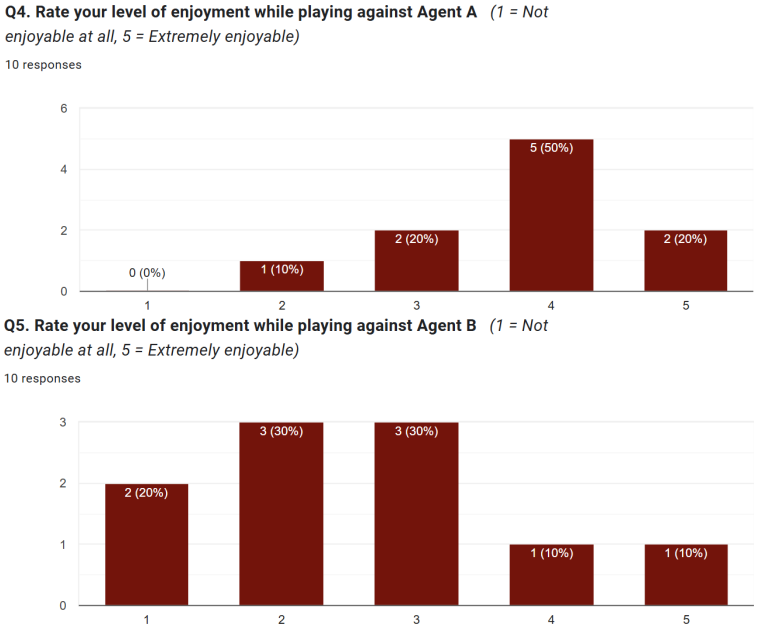

Figure 5. On the top diagram, *Agent A*'s enjoyment results can be seen, while *Agent B* is on the bottom.

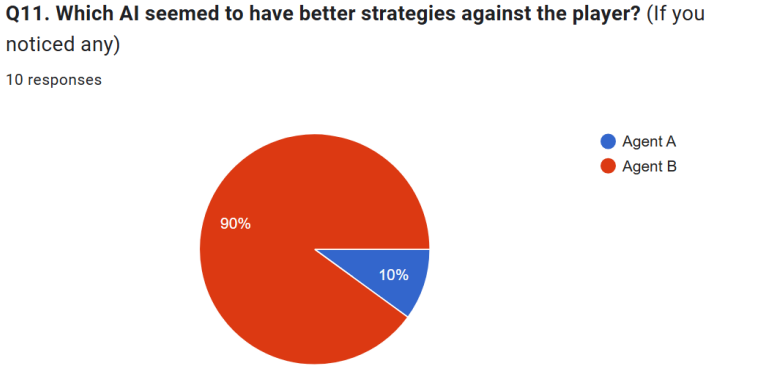

Figure 6. 9 out of ten found Agent B to have better strategies, even though they found him stronger.

1.4.4. Surprise factor in AI tactics

The extent to which the agents surprised players with their tactics was evaluated through Q21. On average, participants rated the surprise factor at 2.4/5, with 4/10 reporting rare surprises, 3/10 noting occasional surprises, and 2/10 indicating frequent surprises. Despite Agent A being perceived as less predictable (Q8), the

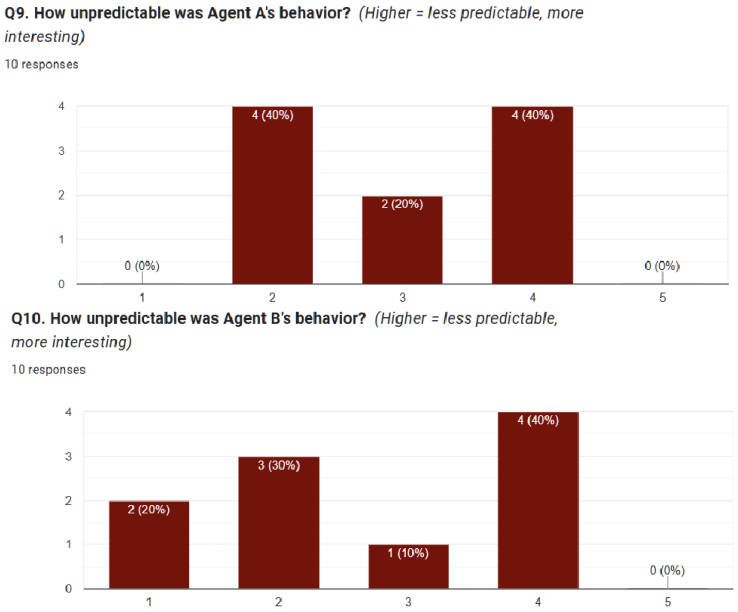

Figure 7. On the top diagram, *Agent A*'s enjoyment results can be seen, while *Agent B* is on the bottom.

moderate surprise ratings suggest that its unpredictability did not consistently translate into novel or innovative tactics. This distinction highlights a potential area for improvement in designing AI behaviors that not only vary in response but also introduce genuinely unexpected strategies to enhance the challenge and replayability of the game.

1.4.5. Impact of reward hacking on perceived unpredictability

Player feedback highlighted a significant issue with Agent B's overuse of the black hole ability, with 60% of participants (Q13) noting that it led to frustrating and repetitive gameplay. One respondent explicitly suggested reward hacking, observing that Agent B used the ability excessively despite designed constraints (e.g., limiting void rifts to two per 10 seconds). This behavior likely contributed to Agent B's lower unpredictability score of 3/5 (Q10) which can be seen in Figure 7 compared to Agent A's 3.2/5, despite its more advanced strategies (Q11). The paradox of reward hacking reducing perceived unpredictability underscores the importance of carefully calibrating the reward system to prevent exploitative behaviors that undermine gameplay variety and player satisfaction.

Overall scores

- **Average gaming experience**: Agent A – 4/5, Agent B – 2.6/5.

- **Frustration**: Agent A – 2.4/5, Agent B – 4/5.
- **Strategic perception**: 70% said Agent B used more advanced strategies, but Agent A was considered more unpredictable.
- **Boss preference**: 7 out of 10 players chose Agent A as their final boss.

Several players criticized Agent B for overusing the "black hole" ability, which made gameplay predictable and sometimes frustrating. In contrast, Agent A had a more varied use of abilities and balanced behavior.

1.4.6. Ultimate preference – Which AI should be the arch enemy?

Most respondents (7 out of 10) selected Agent A as their preferred final boss enemy (Q22), see Figure 8. Agent A was noted for its diverse skill usage, balanced behavior, and lower frustration scores. In contrast, Agent B received more criticism due to its aggressive and often overly dominant use of abilities, particularly the black hole.

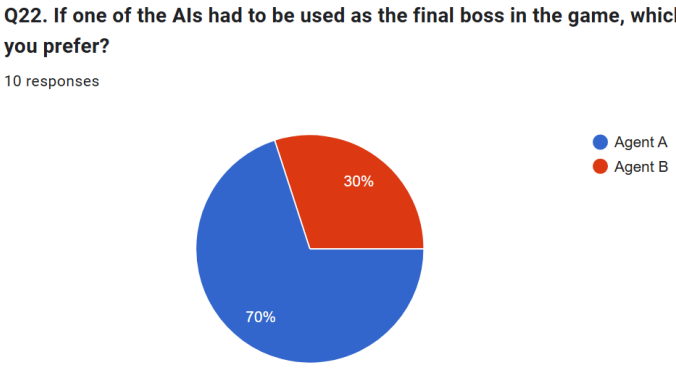

Figure 8. 7 out of 10 people came to the conclusion after answering all the questions and testing both agents that Agent A is their ultimate preference for a game boss.

2. Conclusion

Quantitative analysis using TensorBoard metrics demonstrated that Agent B, trained with initial human guidance, achieved faster convergence, higher ELO ratings, and more stable policy optimization compared to Agent A, which relied solely on self-play. However, qualitative feedback from ten semi-professional players revealed a preference for Agent A, attributed to its greater unpredictability, more varied skill usage, and lower frustration levels (mean: 2.4/5 vs. Agent B's 4/5). Notably, Agent B's overuse of the black hole ability, reported by 60% of participants, not only increased frustration but also paradoxically reduced its perceived unpredictability

(3/5 vs. Agent A's 3.2/5), likely due to reward hacking exploiting the reward system design.

Furthermore, neither agent was consistently perceived as human-like, with Agent B rated slightly higher in realism (3.1/5 vs. 2.6/5), suggesting that advanced strategies alone do not suffice to create the illusion of a human opponent. Additionally, the moderate surprise factor (mean: 2.4/5) indicated that unpredictability did not always translate into novel or innovative tactics, potentially limiting long-term player engagement. These findings underscore the necessity of balancing technical optimization with player experience objectives, prioritizing fairness, tactical variety, and the capacity to surprise players for an engaging gameplay experience.

Future work will explore hybrid training methodologies combining autonomous self-play with periodic human-guided refinement to enhance both strategic competence and perceived realism. Additionally, addressing reward hacking through refined reward structures, such as temporal constraints on ability usage or diversified penalties, will be critical to prevent exploitative behaviors. Approaches like Group Relative Policy Optimization (GRPO)[1] and multi-agent adaptive systems may further ensure that AI opponents remain strategically robust while delivering enjoyable and immersive gameplay experiences.

References

[1] Y. Cheng, Y. Li, Q. Liu, T. Zhang, Y. Zhao, B. Jiang, Y. Zhu, S. Lu, W. Xiong, J. He, et al.: *DeepSeekMath: Teaching Large Language Models to Reason through Latent Programs with Group Relative Policy Optimization*, arXiv preprint arXiv:2402.03300 (2024), doi: 10.48550/arXiv.2402.03300, url: https://arxiv.org/abs/2402.03300.

[2] J. Doran, I. Parberry: *Integrating Rule-Based AI Tools into Mainstream Game Development*, International Journal of Computer Games Technology 2024 (2024), doi: 10.1007/978-3-319-99906-7_23.

[3] R. Erdész: *ARM509_Thesis*, https://github.com/frake92/ARM509_Thesis, Accessed: 2025-08-10, 2025.

[4] A. Juliani, V.-P. Berges, J. Tang, R. Carr, J. Togelius, D. Lange: *Unity: A General Platform for Intelligent Agents*, in: vol. 33, 01, 2019, doi: 10.48550/arXiv.1809.02627.

[5] R. Marcotte, H. J. Hamilton: *Behavior Trees for Modelling Artificial Intelligence in Games: A Tutorial*, The Computer Games Journal 6.3 (Sept. 2017), pp. 171–184, issn: 2052-773X, doi: 10.1007/s40869-017-0040-9.

[6] M. L. Puterman: *Chapter 8 Markov decision processes*, in: Stochastic Models, vol. 2, Handbooks in Operations Research and Management Science, Elsevier, 1990, pp. 331–434, doi: 10.1016/S0927-0507(05)80172-0, url: https://www.sciencedirect.com/science/article/pii/S0927050705801720.

[7] *Quantum Crusade*, https://apestudiosinc.itch.io/quantum-crusade, Accessed: 2025-08-10.

[8] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, O. Klimov: *Proximal Policy Optimization Algorithms*, arXiv preprint arXiv:1707.06347 (2017), doi: 10.48550/arXiv.1707.06347.

[9] D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, A. Bolton, et al.: *Mastering the game of Go without human knowledge*, Nature 550 (2017), doi: 10.1038/nature24270.

[10] R. S. Sutton, A. G. Barto: *Reinforcement Learning: An Introduction*, 2nd, MIT Press, 2018, URL: https://web.stanford.edu/class/psych209/Readings/SuttonBartoIPRLBook2ndEd.pdf.

[11] G. N. Yannakakis, J. Togelius: *Artificial Intelligence and Games*, Springer, 2018, DOI: 10.1007/978-3-319-63519-4.