CURIOSITY-DRIVEN REINFORCEMENT LEARNING FOR AUTOMATED GAME TESTING AND DIFFICULTY ASSESSMENT

by

Atacan Utkusavaş

M.S., Computer Engineering, Boğaziçi University, 2025

Submitted to the Institute for Graduate Studies in
Science and Engineering in partial fulfillment of
the requirements for the degree of
Master of Science

Graduate Program in Computer Engineering

Boğaziçi University

2025

TABLE OF CONTENTS

1. INTRODUCTION

Video games are now large, fast-moving products. Frequent updates, balance changes, and live-service events alter how players move, solve puzzles, and interact with systems. Quality assurance must keep pace with this shifting landscape, but traditional methods fall short. Human playtesters are experienced, yet their time is costly, they tire, and they often repeat familiar strategies. Scripted bots can check routine behaviors, but they follow predefined paths and struggle when the environment changes or when progress depends on creative, multi-step reasoning. As a result, gaps in coverage remain and subtle logic errors can persist until late in development.

This thesis examines automated testing based on agents that learn through interaction, with an emphasis on puzzle and logic-driven games. Instead of relying only on fixed scripts, agents are encouraged to explore and to revisit situations that seem unusual or informative. Two ideas guide the work. The first is curiosity-inspired exploration: agents receive positive signals for reaching unfamiliar or surprising states, which is useful when clear rewards are rare or delayed. The second is coordinated exploration: several agents run in parallel and are gently steered to avoid repeating each other's work. Together, these ideas aim to create a testing process that adapts to changing content, scales across levels, and reveals problems that rigid scripts may miss.

The goal is not only broader coverage but also clearer feedback for designers. Runs are recorded and summarized so that teams can see where agents spent time, which interactions led to dead ends, and how the overall behavior changes across builds. From these summaries, the system produces simple, comparable indicators—such as where difficulty spikes occur or where progression becomes unstable—so that changes in a level can be judged quickly and consistently.

1.1. Motivation

Modern production cycles demand testing that remains useful while content changes rapidly. Puzzle games make the need especially visible: small edits to switches, keys, timers, or ordering rules can make a level trivial, impossible, or exploitable. Human testers can uncover many such issues, but the space of possible action sequences grows quickly, and rare failures often depend on long chains of steps that are difficult to discover and reproduce.

An exploration-driven, agent-based approach offers a practical way forward. By rewarding agents for reaching unfamiliar states, the system increases the chance of encountering the unusual situations that matter for QA. Running several agents at once, with light coordination, helps cover more of the level in less time and reduces redundant paths. When agent behavior is captured carefully, the same sequences can be replayed later, giving designers stable evidence of how a failure occurred and whether it still occurs after a change.

The motivation is therefore straightforward: teams need an automated tester that keeps up with content updates, finds non-obvious problems, and turns raw play into concise, comparable summaries that guide design decisions. Curiosity-inspired exploration, multi-agent coverage, and clear reporting are combined here to reduce manual effort while improving both breadth of testing and the usefulness of results.

1.2. Problem

Bringing learning-based agents into real QA introduces several challenges. Feedback is often sparse: meaningful signals may appear only after many steps, which can make learning slow or unstable. Agents can also gravitate toward familiar parts of a level and miss deeper regions that matter for coverage. When multiple agents are used, naive parallelism can waste resources on similar trajectories unless there is some mechanism to encourage diversity. In addition, strong results on one level do not au-

tomatically transfer to others, so the approach must minimize per-level tuning and remain robust as content evolves.

There is also a mismatch between the metrics that help train agents and the information designers need. Internal scores and returns do not directly answer questions such as whether soft locks exist, which interaction chain causes them, how often they can be reproduced, or how a tweak shifts difficulty. Without appropriate logging and aggregation, automated testing appears opaque and is hard to act on.

This thesis addresses these issues with a general framework shaped around three layers. First, agents are guided by simple signals that favor novel or informative states, without assuming any particular curiosity model. Second, exploration is run in parallel, with a light mechanism to reduce overlap so that agents collectively cover more of the space. Third, a reporting layer converts raw trajectories into developer-facing artifacts: concise coverage summaries, reproducible failure traces, and difficulty indicators derived from how behavior evolves over time. Put simply, the problem is to build an automated tester for puzzle-like games that explores broadly, scales across changing content, and produces evidence that integrates naturally into everyday workflows. The rest of the thesis describes the design choices, evaluation setup, and findings that support this aim.

2. RELATED WORKS

Research on using Reinforcement Learning (RL) for automated game testing has grown steadily in recent years. As games became larger and more dynamic, traditional scripted testing methods began to fall short. Several studies explored how learning-based agents can support quality assurance by discovering unexpected states, interacting with complex systems, and identifying logic issues that might be missed by human testers. In this section, we summarize four representative works that shaped the direction of RL-based game testing. Each of these studies approaches the testing problem from a different angle, but they all share the goal of improving exploration, coverage, and detection of unusual behaviors in interactive environments.

The first line of work focuses on curiosity-driven reinforcement learning. Ferdous et al. [1] introduced a system in which an agent is motivated to explore by rewarding itself for encountering unfamiliar or surprising game states. Instead of following a fixed checklist of objectives, the agent learns by interacting with the environment and observing how its actions influence future states. This curiosity-driven design helps the agent reach areas that would normally require long or unintuitive action sequences. The study demonstrated that intrinsic motivation can significantly enhance exploration, making it possible to uncover hidden rooms, unreachable objects, and rare logic errors that scripted bots often miss. Their results showed that curiosity can act as a general driver of exploration across different levels without needing carefully handcrafted reward functions.

A later extension by the same authors expanded this idea into a multi-agent setting. In this follow-up study, Ferdous et al. [2] examined how curiosity can be scaled across multiple agents exploring the same 3D environment at the same time. Each agent still follows its own curiosity signals, but they also receive a light coordination signal that encourages them to avoid repeating each other's behavior. This prevents redundant exploration and increases the overall coverage of the environment. The

study found that multi-agent cooperation leads to faster discovery of new areas and more efficient exploration. It also showed that the combined behavior of several curious agents can reveal multi-step issues in larger maps that a single agent may struggle to find alone. The work highlights the importance of diversity when the goal is broad and thorough testing.

A third line of work comes directly from the game industry. Bergdahl et al. [3] presented one of the first attempts to integrate deep RL into professional game testing pipelines. Working within Electronic Arts (EA), the authors built a system that connects game engines to RL agents capable of navigating levels, interacting with objects, and attempting gameplay objectives. Their study emphasized the practical challenges of using RL in real production environments, such as the need for stable training, reliable logging, and interpretable outputs. The agents were used for tasks like navigation testing, collision detection, difficulty measurement, and regression tracking across different builds. The authors argued that RL does not replace human testers but acts as a consistent and scalable assistant that can explore widely and provide reproducible evidence of unexpected behavior.

This work also demonstrated how important it is to integrate RL systems with existing development tools. The authors showed that agents need structured access to game state information and reliable ways to record what they encounter. Their system made it possible to compare different builds of the same level and observe how small changes affected agent behavior. This is particularly useful for confirming bug fixes or identifying situations where progression unintentionally becomes easier or harder. The study illustrates that even relatively simple RL approaches can contribute meaningfully to quality assurance when embedded properly into a production workflow.

Another influential contribution comes from Zheng et al. with their Wuji framework [4]. Instead of relying solely on reinforcement learning, Wuji combines deep RL with evolutionary optimization. In this system, a population of agents with different strategies explores the environment. Over time, better-performing agents are selected

and modified through evolutionary operations. This design promotes behavioral diversity and helps the system avoid getting stuck in repetitive or unproductive patterns. The study showed that evolutionary search can support broad exploration and uncover unusual issues, especially in games where rewards are sparse or incomplete.

The strengths of Wuji lie in its ability to maintain a diverse set of agents and search for unexpected interactions. Because each agent follows a slightly different strategy, the system as a whole is more likely to find edge cases or physics-based anomalies. The study demonstrated that evolutionary algorithms can complement RL by guiding exploration toward states that might otherwise go unnoticed. Although this approach requires more computational effort, it shows that diversity-driven search can be a powerful tool for testing large and dynamic game worlds.

Across all four works, a common theme emerges: effective automated testing requires agents that can explore beyond what scripted rules allow. Curiosity-based signals, multi-agent coordination, industrial integration, and evolutionary search each provide a different way to achieve this goal. Together, they demonstrate that RL can help uncover complex behaviors, identify hidden logic issues, and produce meaningful traces that designers can act on. These studies also highlight the gap between controlled research environments and real production workflows, showing that a practical approach must balance learning stability, exploration diversity, and interpretability.

Overall, the literature suggests that RL offers a promising foundation for automated game testing, especially when exploration is essential. Curiosity, cooperation, and behavioral diversity consistently appear as useful ingredients for discovering rare or non-intuitive states. At the same time, real-world adoption requires clear reporting, reproducible outputs, and tight integration with existing tools. The work reviewed here provides motivation for developing a system that brings these ideas together in a form suitable for testing puzzle and logic-based games, which offer structured environments where exploration-driven methods can provide immediate benefits.

3. METHODOLOGY

This chapter presents the methodology for a reinforcement-learning-based framework that this thesis proposal aims to develop for automated game testing and difficulty assessment. The goal is to design a general and reusable approach that will operate across multiple game genres while remaining practical and informative for puzzle games, which serve as accessible and controlled test environments. The framework is shaped by three recurring themes in both the literature and industry discussions: firstly, reinforcement learning (RL) agents can act as scalable game testers capable of exploring far more of a game than a human can; secondly, curiosity-driven exploration would help agents reach unusual or non-intuitive states that often contain hidden design issues; and lastly, developers will require interpretable and reproducible outputs—not just reward curves—to understand level difficulty, progression stability, player diversity, and failure modes. The result will be a modular system that trains agents, coordinates multiple explorers, adapts their behavior to simulate different types of players, records their trajectories, and transforms those traces into developer-facing metrics that reflect both coverage and difficulty across different player skill profiles.

3.1. Overview of the Framework

The proposed framework is built around three interconnected layers. The first is the learning layer, where agents will interact with a game environment and optimize a policy that encourages robust and diverse exploration. The second is the coordination layer, which will manage multiple agents so that they do not follow identical paths but instead complement one an others behavior. The third is the analysis layer, which aims to examine the learned behavior and convert it into metrics that designers can use before releasing a level. All three layers will integrate a mechanism for modeling different types of players—such as “casual,” “average,” and “expert”, so that the same level can be evaluated from multiple perspectives.

The process will begin by extracting a structured observation from the game. In puzzle games, this may include the board configuration, remaining moves, object locations, or interaction states. In other genres, observations might include physics states, enemy positions, cooldown timers, or map coordinates. Keeping observations general will allow the framework to connect to different game engines with minimal engineering work.

Each agent will receive two kinds of reward: extrinsic rewards that correspond to the game's objectives, and intrinsic curiosity-based rewards that encourage exploration of unfamiliar or unpredictable states. This will be especially useful in sparse-reward environments—such as multi-step puzzles—where reaching the goal may require a long chain of correct actions. A parallel mechanism will modify observations, action precision, and exploration behavior to simulate different human capabilities. Over time, an agent is expected to uncover intended solutions, shortcuts, soft locks, and bugs that traditional scripted tests would not capture. A high-level overview of the system is shown in Figure 3.1.

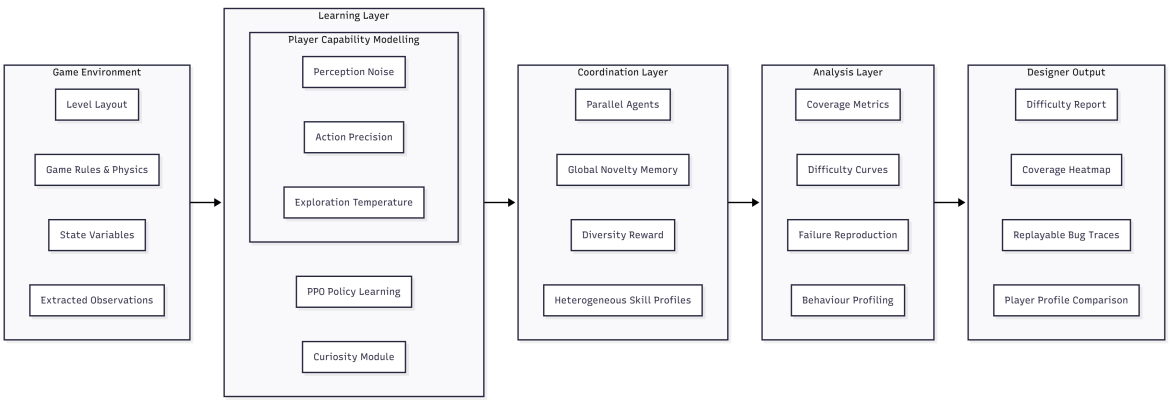

Figure 3.1. Overview of the proposed framework

3.2. Learning Layer

The learning layer will be responsible for training curiosity-driven RL agents. This proposal plans to use Proximal Policy Optimization (PPO), as it generally performs reliably in environments that change frequently during development and because it balances stable learning with the ability to adapt.

At every step, an agent will receive an observation, select an action, and observe the outcome. The reward it receives will be a combination of the game's own scoring system and an additional curiosity bonus that would encourage it to seek out unpredictable or unexplored states. The curiosity module will learn to predict what the next state should look like. Whenever the game behaves differently than expected, the agent will perceive that as novelty and receive an intrinsic reward for investigating the area further.

3.2.1. Player Capability Modeling

Real players vary widely in how they perceive the game, how precisely they act, and how consistently they follow an intended solution path. To reflect this diversity, the learning layer will include a mechanism that modifies the agent's behavior along three dimensions:

- **Perception quality:** Casual or inattentive players may overlook certain details in a level. This behavior will be reproduced by adding controlled noise or masking parts of the observation. Expert players will receive clear and accurate information.
- **Action precision:** Less experienced players act with more randomness or hesitation. This will be simulated by increasing randomness in the agent's action choices. Expert agents would act more decisively and consistently.
- **Exploration style:** Some players explore freely, while others play with purpose. Adjusting the agent's exploration enthusiasm will allow it to behave more like a confused beginner or a confident experienced player.

3.2.2. Game Modeling Across Genres

To support different types of games, each environment will be described using a consistent template that includes its state representation, available actions, transition rules, reward structure, termination conditions, and the chosen player profile. Below

are examples based on real game titles that represent their genres:

- **Wiggle Escape (Path-planning Puzzle):** The state will include the grid layout, the snake's position and direction, obstacles, and collectibles. Actions will consist of four directional moves. Useful metrics will include steps required to escape, frequency of dead-ends, probability of soft-locks, and a measure of action hesitation.
- **Merge Studio: Fashion Makeover (Merge-Logic Game):** The state will include the inventory, merge graph, and current objective. Actions will include merging pairs of items, discarding unneeded items, and claiming produced items. Metrics will include the depth of merge chains explored, number of steps to reach a target, frequency of irreversible mistakes, and discovery rate of rare or obscure merge paths.
- **Tile Star: Match Puzzle Games (Match-3 Puzzle):** The state will include the tile grid, tile types, special tiles, and remaining moves. Metrics will include probability of solving the level, move efficiency, how well the difficulty curve aligns with move constraints, and the diversity of tile configurations encountered by the agents.

3.3. Coordination Layer

Running a single agent will reveal important problems, but running many agents in parallel would produce significantly broader coverage. The challenge is that multiple agents often end up repeating similar behavior. To avoid this, the coordination layer will include a lightweight mechanism that tracks which regions of the game have already been visited. Agents will be encouraged to explore parts of the level that have been visited less often. This is intended to lead to better coverage of the state space without requiring complicated communication between agents.

This layer will also support heterogeneous player profiles running simultaneously. For example, one agent may represent a distracted novice while another represents an

experienced or skilled player. Comparing their paths would reveal whether a level is too punishing for beginners or unexpectedly trivial for experts.

3.4. Difficulty and Behavior Analysis Layer

Once training is completed, the agents will be used to generate rollout trajectories that show how they behave when attempting to solve the level. These trajectories will serve as the foundation for several key analysis metrics:

- **Coverage:** Coverage will measure how much of the environment the agents actually explored. Low coverage may indicate bottlenecks, overly strict level flow, or hidden states that are difficult to reach. High coverage suggests that the agents examined many possible states and interactions.
- **Failure reproducibility:** If an agent encounters a soft-lock or an unintended loop, the system will record the entire sequence along with the simulation seed. Designers will be able to replay the failure exactly to inspect the underlying issue.
- **Difficulty indicators:** Difficulty will be estimated by examining how quickly agents of different skill profiles learn the level, how often they succeed once trained, and how sensitive they are to small disturbances. Comparing these indicators will help identify levels that may be too easy, too difficult, or too unstable.
- **Additional behavior patterns:** These include variability of successful trajectories, deviation from the shortest known solution, and sensitivity to small observation changes.

3.5. Integration and Workflow

The methodology is intended to run continuously as part of the development cycle. When a designer makes a change to a level, the framework will retrain or fine-tune the agents, collect new trajectories, and update the difficulty and coverage metrics. If a change accidentally makes the level unfair for beginners or removes all challenge for

advanced players, the system would highlight the issue.

Although puzzle games are the easiest place to begin experimenting with this approach, the framework is expected to generalize to other genres that share similar interaction patterns. This would be particularly useful for studios maintaining multiple related games.

3.6. Conclusion

This proposal aims to design a practical and extensible reinforcement-learning-based framework for automated game testing and difficulty assessment. The system will combine curiosity-driven exploration, multi-agent coordination, and structured behavioral analysis to help uncover unusual states, detect soft-locks, and provide reproducible evidence of level behavior. By modeling different player capabilities, the framework is intended to offer designers a clearer understanding of how difficulty changes across skill profiles. Puzzle games will serve as an initial controlled setting, but the approach is designed to generalize to other genres without extensive re-engineering.

The expected contribution of this proposed work is a unified testing pipeline that supports broad exploration, interpretable outputs, and integration with real development workflows. The project aims to demonstrate that learning-based agents can act as scalable and reliable testing tools, especially in environments where manual or scripted approaches fall short. As the next step, the implementation phase will begin with small open-source puzzle environments to validate the feasibility of curiosity-driven RL and multi-agent coordination before expanding toward more complex game structures.

4. THESIS SCHEDULE

Thesis schedule timeline is as follows:

Timeline	**Task Description**
January 2026 - February 2026	Final Literature Review & Scope Refinement
February 2026 - June 2026	Environment Modeling & RL Pipeline Development
June 2026 - September 2026	Experimental Phase & Testing on Games
September 2026 - October 2026	Analysis & Interpretation
October 2026 - November 2026	Thesis Writing & Defense Preparation

REFERENCES

1. H. Ferdous, B. Marin, A. Iglesias, and T. Vos, "Curiosity-driven reinforcement learning for 3d game testing," in *Proceedings of the 37th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 2022, pp. 1142–1153.

2. H. Ferdous, T. Vos, B. Marin, and A. Iglesias, "Curiosity-driven multi-agent reinforcement learning for 3d game testing," in *Proceedings of the 38th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. arXiv preprint arXiv:2502.14606, 2025. [Online]. Available: https://arxiv.org/abs/2502.14606

3. J. Bergdahl, C. Gordillo, K. Tollmar, and L. Gisslén, "Augmenting automated game testing with deep reinforcement learning," in *2020 IEEE Conference on Games (CoG)*. IEEE, 2020, pp. 497–504.

4. Y. Zheng, X. Xie, T. Su, L. Ma, J. Hao, Z. Meng, Y. Liu, R. Shen, Y. Chen, and C. Fan, "Wuji: Automatic online combat game testing using evolutionary deep reinforcement learning," in *Proceedings of the 34th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 2019, pp. 772–784.