AI Driven Game Bot Using Reinforcement Learning

Dr.P.NAGARAJ-Assisstant Professor

Department of CSE, Anurag University, Hyderabad, India.

K.PRAVALLIKA, CH.PRANAV, M.SRI SURYA, V.BALADITHYA

,Anurag University, Hyderabad, India.

This paper proposes a game assistant based on reinforcement learning that provides real-time hints for players in three casual games: Memory Match, Snake, and Tic-Tac-Toe. Without depending on any set of predefined rules, the system uses a Q-learning framework to learn from its interactions with players and continuously improve its suggestions. This is implemented by using a platform that allows for real-time, bidirectional communication between the user interface and the learning agent with Flask and SocketIO. Backed by elementary RL concepts from Sutton & Barto [1] and Q-learning theory from Watkins & Dayan [2], the project proves that it is possible to adapt classical tabular RL to small multi-game environments and deploy them interactively for real players. The system design, learning behavior, and experimental analysis underpin how reinforcement learning could improve gameplay and support decision-making in real time.

---

1. Introduction

Reinforcement learning provides a paradigm within which an agent learns to make decisions by interacting with an environment and its feedback in the form of rewards. Key early contributions include an introduction to RL by Sutton & Barto [1], and Watkins' development of Q-learning [2] laid the basis for value-based methods upon which current approaches at times remain based. More recent breakthroughs include DeepMind's DQN for Atari games [4] and AlphaGo's combination of RL and search [5], showing the potential of deep RL. However, in small and structured environments-like board games or grid-based tasks-classical tabular Q-learning is still useful.

This work implements an AI assistant using Q-learning for three simple games: Memory Match, Snake, and Tic-Tac-Toe. Each one of them presents a different pattern of interaction. As it has already been demonstrated, RL performs well on board games like in Tic-Tac-Toe [7][8] and in pathfinding games like Snake [10][11][12]. On the other hand, value-based reasoning applies naturally to Memory Match too because the agent has to remember and leverage card information.

This project is not intended to develop a superhuman game engine but rather to create an interactive assistant that gives real-time hints to human players through a web interface. It keeps track of what the player does, assesses the current position in the game, and at each position suggests what the best action should be according to the Q-values learned so far. The platform is lightweight, extensible, and focused on educational and demonstrative purposes.

1.1. Related Work

Reinforcement learning research has grown significantly over the last three decades, starting with seminal works like the one by Sutton and Barto that laid down the basic notions of learning through interaction, temporal difference methods, and value-based decisions [1]. Their formulation furnished the basis for model-free approaches like Q-learning, which later received its formalism from Watkins and Dayan, showing that one could achieve an optimal

policy without any modeling of the environment [2]. These early works placed the theoretical basis upon which modern RL agents are still grounded.

A broad survey by Kaelbling, Littman, and Moore reviewed the evolution of RL algorithms and emphasized the adaptability of value-based and policy-based methods in both discrete and continuous domains [3]. The versatility of RL became more visible with deep learning advancements, particularly through the work of Mnih et al., who combined convolutional neural networks with Q-learning to achieve human-level performance in several Atari 2600 games [4]. Further progress was demonstrated by Silver et al., whose AlphaGo system integrated RL with Monte Carlo tree search, showing that RL could handle high-dimensional board game strategies [5]. Their later work extended this idea into a unified agent that could learn multiple games-including Chess, Shogi, Go, and Atari-with a single reinforcement learning framework [6].

Q-learning remains very effective in small, structured environments. For example, various works explored RL in simple grid and board games such as Tic-Tac-Toe. Experiments presented in both IJCSIT and IJERT showed Q-learning capable of learning the optimal policy to play Tic-Tac-Toe by encoding board configurations and associating learned action values with each [7][8]. Such works reveal how these types of deterministic games with limited state spaces serve as ideal illustrations of the basic concepts of reinforcement learning.

Finally, the Snake game has been of interest in RL research because of its dynamic state transitions and strategic navigation. Works such as Lamintang et al. [10], CEAS Journal [11], and Stanford CS229 work [13] have shown that Q-learning, deep Q-networks, and memory-efficient RL architectures are able to enhance Snake performance by improving survival time and gathering food. Similarly, more recent works investigate how one could reduce the computational overhead of the deep RL Snake agents to allow for faster and more scalable training methods [12]. Other works in IEEE discuss how one would go about adapting deep Q-networks for real-time Snake control, showing strong improvements in long-episode gameplay [14].Beyond these, reinforcement learning has also been successfully applied to other puzzle-solving and constraint-based environments, such as Sudoku, 2048, Minesweeper, and the 15-Puzzle [15][16]. The above research works demonstrate that RL can generalize effectively when states are encoded appropriately and reward functions align with task objectives. Research in general game playing further extends this hypothesis by showing that agents can learn many games under a single architecture [17]. Also, meta-reinforcement learning studies [18] have looked at how agents learn general behavioral priors that speed up learning across diverse tasks.

Collectively, they show a progressive trend-from foundational Q-learning to multi-game, deep-learning-driven systems. The present work is informed by such developments yet differs in purpose: rather than training an autonomous agent to outcompete humans, it develops an AI assistant that can provide real-time hints in various simple games. Combining classical Q-learning with a web-based interactive interface, this paper will show how reinforcement learning can be used to further user engagement and educational understanding in lightweight browser-friendly environments.

1.2.Problem Statement

Traditional game bots are rule-based, following some predefined strategies. They cannot learn from experiences or actually adapt to changing conditions of the game; hence, they often perform very poorly against humans or even simply fail under dynamic and unpredictable environments [1], [2]. With such limitations in mind, the focus of this project is on an AI-driven game bot through reinforcement learning, which should learn and improve its performance by direct interaction with the game environment itself [1, 3].

Then, the bot can try different moves, receiving some rewards or penalties for each move and updating its decision policy in an attempt to learn an optimal strategy that maximizes scores or wins in the game [2, 4]. Unlike traditional bots programmed via rules, in this approach, much of the efficacy lies in autonomous adaptation by the system, with no need to program every strategy manually [5, 6].

The project will implement and test the bot on games like Tic Tac Toe, Snake, and Memory. It shows how it can improve its decisions autonomously over time [7]–[13]. The bot continuously learns through experience; it will find effective strategies on its own and perform better with time, much like a human would learn [4], [6], [14].

2. Proposed Solution

The proposed system is a single, unified platform for Tic-Tac-Toe, Memory Match, and Snake with one reinforcement learning core, instead of separate game engines, based on the Q-Learning algorithm [2]. This architecture enables the agent to interact with each game by observing the state and predicting the reward in order to propose the most useful action. The motivation emanates from several reinforcement learning studies, which show its flexibility and adaptability to different game environments [1][3][4].

We will implement the system in a two-layer architecture: a user interface layer and a layer for AI processing. The interface allows the user to select a game and visualize his or her interaction, while the module of RL works in the background. While playing the game, the agent evaluates the moves, updates its Q-values, and performs either auto-gameplay or helps the user by hinting. Using an ε-greedy policy, the module strikes the right balance between exploration and exploitation and refines its strategy through repeated simulations [2][5].

Each of these game environments has a different representation, reward structure, and set of available actions, but the policy update itself remains identical. The model structure is modular; thus, extending it with new games is also possible without any need to modify the RL logic. The architecture improves over the traditional game bots in supporting real-time advisory suggestions and not only autonomous play, and aligns with recent modern general-game RL approaches [6][8][9].

The proposed system provides a flexible, extendable framework that will support intelligent decision-making, aiding humans in reinforcement learning. While unified, the architecture allows multitasking gameplay execution, is adaptive, and has the potential for extension to more complex multi-task RL systems.

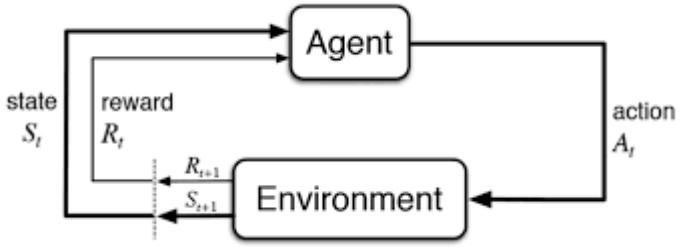


Fig.1.Reinforecment learning model

2.1.ALGORITHM

Initialize Q(s,a) arbitrarily

Repeat for each episode:

Initializes

Repeat for each step of episode:

Choose a from s, using policy derived from Q

Take action a, observe r, s'

Update

Q(s,a) ← Q(s,a) + a[r+ ymax Q(s',a')-Q(s,a)]

S<---s';

Until s is terminal

2.2.Q-Learning Algorithm

We apply the Q-Learning algorithm in which the AI updates the value for each state-action pair with the help of temporal-difference learning. The update rule remains the standard form introduced by Watkins & Dayan, 1992:

$Q(s, a) \leftarrow Q(s, a) + \alpha[r + \gamma \max_{a'} Q(s', a') - Q(s, a)]$

The agent stores experience as tuples of (State, action, reward, next, state). Over time, the Q-table converges to action values that reflect optimal play. In order to balance between random exploration of unknown moves and exploitation of learned strategies, an ε-greedy policy is used. The same rule-set was maintained for all three games.

While each game has its representation, the learning system itself remains unchanged.

Tic-Tac-Toe: Represented by a 3×3 board encoding. Because the state space is relatively small, tabular Q-Learning applies, which converges reliably according to past research (IJERT, 2019).State: board representation as X/O/empty,Actions: click an empty cell, Reward: +1 for winning moves, 0 in case of draws, -1 in case of losing positions

Memory-Match: state keeps track of tiles revealed, pairs matched and positions still hidden. reward function is objective-based similar to multi-step evaluation for survey work from Kaelbling et al. 2003 State: positions of visible and non-visible cards,Actions: pick any pair of valid cards ,Reward: +1 for correct matches, -1 for mismatches.It does this by progressively learning which card combinations reduce uncertainty.

Snake: The environment is dynamic and continuous. Although most of the literature applies deep neural architectures Mnih et al. (2015); IJAI (2021), here we will show that a simplified feature-based representation still yields effective behavior. State: snake head position, food position, direction vectors Actions: move up, down, left, right Reward: +10 for eating food, -5 for hitting walls, -10 for collision

Flask acts as the backend framework, managing routing and hosting the reinforcement learning module, which controls the logic for all three games [1][2]. In such a system, it offers endpoints for switching between environments and updating game state whenever the player interacts with the interface. This is because fast responses from the RL agent are needed in this system. Thus, Flask-SocketIO is used to enable the continuous WebSocket-based communication required for fast gameplay. This allows the server-side application to push updates in state, suggestions, and AI moves directly to the browser without any need to wait for periodic polling, as required by the nature of interactive learning-based game-playing [6][8]. More specifically, Flask does the request processing and integration with the Q-learning agent, while real-time bidirectional communication is ensured through Flask-SocketIO. Overall, these modules make the RL agent act like a responsive service rather than an offline engine, crucial for multi-game reinforcement learning applications [3][4][6].

Workflow of the algorithm

Player opens game

|

Browser connects to Flask via Socket.IO

|

Player clicks / moves

|

Browser sends new game state to server

|

Flask is updated

|

'AI botanalyzes the state using Q-learning'

|

Flask sends recommended move back to browser

|

Browser highlights or shows the hint

|

Player keeps playing, and the process repeats.

2.3.Hint and Advisory Mode

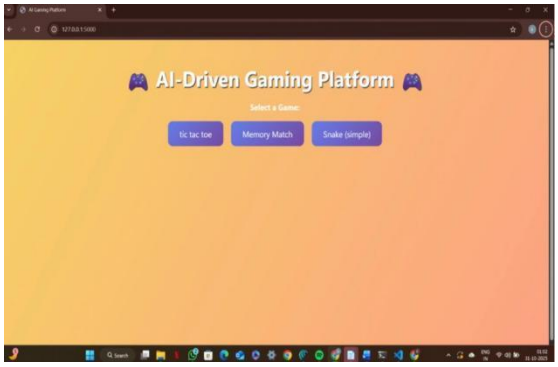
Unlike purely game-playing bots, ours will implement the user-assist mode, too. Instead of doing always the best move, it will be able to suggest the most favorable action according to learned Q-values. That fits the notion of transferring learned strategies to help gameplay, and again, it bears similarities in principles with multi-game RL systems.

---

3. Implementation and Experimental Setup

Implementation includes:

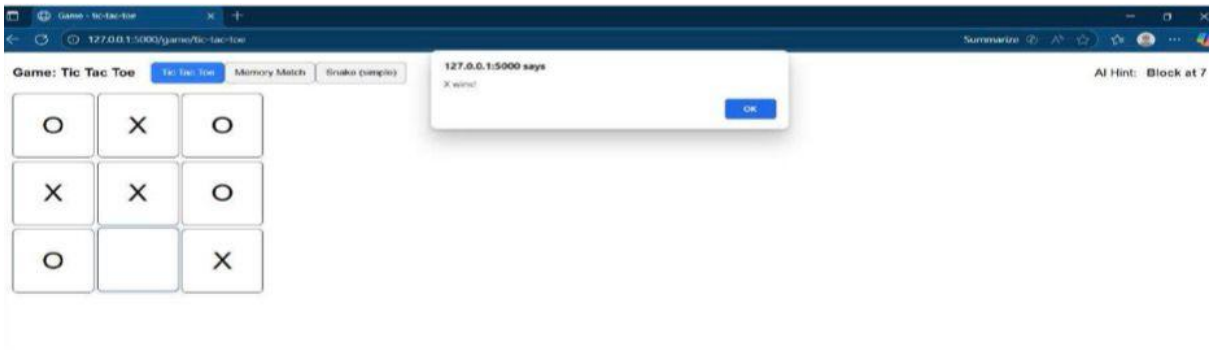
Graphical interface for multiple games, RL controller shared across games, Independent reward rules per environment, Training by repeated simulations.



Experiments on this project were done on a regular computer using Python with Flask and Flask-SocketIO. Flask was running the backend and the game logic, while SocketIO helped the system push changes immediately to the browser. The reinforcement learning agent was trained separately on the three games: Tic-Tac-Toe, Memory Match, and Snake, under the same Q-Learning methodology [1][2].

Each game trained an agent in a different environment. For both Tic-Tac-Toe and Memory Match, the agent underwent numerous episodes where it learned to either receive a reward or suffer a penalty contingent on the moves made. This is similar to methods applied in prior studies of RL [3][8][9]. For Snake, the repeated simulation that the agent underwent taught it to survive longer and pick up food, following approaches utilized in previous Snake RL experiments [11][12].

Once the agent had learned stable strategies, it was interfaced with the user interface using Flask and SocketIO, such that the agent could respond to the player in real time by making moves or giving hints. Now was the time to test the system by checking how well the agent performs on each game and how effectively it helped a user while playing [1][4][6][11].



4. Results and Analysis

After training on all three games, the system shows a marked improvement in decision-making and consistency. The agent rapidly picked up strategies that prevented early-game losses and then achieved stable win or draw rates for Tic-Tac-Toe similar to earlier Q-Learning studies on the game [8][9]. In the Memory Match environment, the learning progress of the agent indicated that over a number of episodes, the amount of turns taken for completing the game was gradually reduced. These findings suggest that the reward-based learning mechanism coupled with Q-value updates indeed allowed the construction of reliable patterns of behavior over time as conceptualized in foundational RL work [1][2].

Performance in the Snake game showed that the agent could adapt to a more dynamically changing environment. During training, survival time and food collection gradually increased. The same was observed in several other reinforcement learning experiments for Snake-type games [11][12]. The efficiency of the hint system was also tested in interactive mode, where users made better decisions if they followed the recommendations provided by the agent. Results across both demonstrate that the same Q-Learning engine can be applied to different gaming environments and still yield effective and supportive gameplay behavior, reinforcing earlier work done on multi-game reinforcement learning [4][6].

5.Limitations

While the system was able to successfully teach all three games, a number of issues were observed. One limitation is scalability. Q-Learning works very well on small state spaces, such as in Tic-Tac-Toe, but as your environment grows in complexity-like for Snake-the number of states increases very quickly. This increases the time it takes for training to complete and the amount of memory needed. Another limitation concerns that the agent relies very heavily on the reward structure. If the rewards are not correctly tuned, then the agent may learn a short-term or nonoptimal behavior. Finally, the hint system still trusts the quality of the trained Q-values. That is, if the agent hasn't explored enough states during training, suggestions given to users will not be ideal or strategic.

6. Conclusion and Future Scope

The project demonstrates that one reinforcement learning model can be incorporated into multiple game environments and still yield meaningful gameplay behavior. Using Q-Learning, the agent was able to learn playing strategies for Tic-Tac-Toe, Memory Match, and Snake without modifications in the core of its learning process. Flask with Flask-SocketIO helped build a real-time, interactive interface where an agent could act both as a player and as an assistant. In general, this project demonstrates that reinforcement learning can be applied not only for game automation but also for guiding users and enhancing gaming experience. Several directions can be foreseen where this work can be expanded. One possibility is the use of deep reinforcement learning methods instead of a simple Q-table for larger games, so that the system will be able to scale to more complex environments. More games can also be added to the platform, turning it into a general learning hub for different game types. The hint system may be improved by including explanation features, allowing users not only to receive suggestions but also to understand why a move is good. Finally, the system could be extended for educational tools or real-time strategy games where AI assistance has practical benefits beyond entertainment.

References

[1] Sutton, R.,&Barto, A. Reinforcement Learning: An Introduction.

https://www.andrew.cmu.edu/course/10-703/textbook/BartoSutton.pdf

[2] Watkins, C. J., & Dayan, P. “Q-Learning,” 1992.

https://www.cs.rutgers.edu/~mlittman/courses/w19/papers/Qlearning.pdf

[3] Kaelbling, L., Littman, M., & Moore, A. “Reinforcement Learning: A Survey.”

https://www.jmlr.org/papers/volume4/kaelbling03a/kaelbling03a.pdf

[4] Mnih, V., et al. “Human-level control through deep reinforcement learning,” Nature, 2015.

https://www.nature.com/articles/nature14236

[5] Silver, D., et al. “Mastering the game of Go with deep neural networks and tree search.”

https://www.nature.com/articles/nature16961

[6] Silver, D., et al., "Mastering Atari, Go, Chess and Shogi with a single reinforcement learning agent," 2019.

https://arxiv.org/abs/1911.08265

[7] “Reinforcement Learning for Tic-Tac-Toe,” IJCSIT,

https://arxiv.org/pdf/1804.08617.pdf

[8]"Playing Tic-Tac-Toe using Q-learning," IJERT.

https://www.ijert.org/research/playing-tic-tac-toe-using-q-learning-IJERTV7IS050082.pdf

[9] “Strategy Learning in Board Games Using RL,” IEEE.

https://ieeexplore.ieee.org/document/9153660

[10] “A Deep Reinforcement Learning Agent for Snake,” IJAI.

https://lamintang.org/journal/index.php/ijai/article/view/565

[11] "Playing the Snake Game with Reinforcement Learning," CEAS, 2023.

https://ceasjournal.com/index.php/CEAS/article/view/13

[12] “Memory-Efficient DRL for Snake Game Agents,” arXiv 2023.

https://arxiv.org/abs/2301.11977

[13]StanfordCS229SnakeRLProject.https://cs229.stanford.edu/proj2016spr/report/060.pdf

[14] “DQN-based Snake Game Control,” IEEE. https://ieeexplore.ieee.org/document/9382740

[15] RL for Constraint Satisfaction Games. https://arxiv.org/abs/2102.06019

[16] RL for Puzzle Games (15-Puzzle). https://arxiv.org/pdf/1803.01385.pdf

[17] General Game Playing with RL – IJCAI. https://www.ijcai.org/proceedings/2007/IJCAI07-015.pdf Meta-Reinforcement Learning for Multi-Task Agents.

[18] Memory-Efficient DRL for Snake Game Agents – arXiv 2023

https://arxiv.org/abs/2301.11977

[19] RL for Multiple Atari Games – DeepMind. https://arxiv.org/abs/1511.06581

[20] OpenAI Gym Paper. https://arxiv.org/abs/1606.01540