Rotating Blocks: A Visual Tetris Puzzle Game

Prabhav Ravi Tammanashastri
Dept. of Computer Science Engineering
Amrita School of Computing,Bengaluru
Amrita Vishwa Vidyapeetam,India
BL.EN.U4CSE21162

Suhas S Bhat
Dept. of Computer Science Engineering
Amrita School of Computing,Bengaluru
Amrita Vishwa Vidyapeetam,India
BL.EN.U4CSE21200

Vijayakumar Soorya
Dept. of Computer Science Engineering
Amrita School of Computing,Bengaluru
Amrita Vishwa Vidyapeetam,India
BL.EN.U4CSE21220

***Abstract*—This project presents a unique implementation of the classic Tetris game integrated with advanced machine learning capabilities using Unity and the Unity ML-Agents Toolkit. The project employs the Unity ML-Agents Toolkit to incorporate AI-based gameplay mechanics, allowing non-player agents to learn and develop optimal strategies for arranging blocks within the environment. It trains intelligent agents which are able to solve these puzzles or compete against human players using reinforcement learning. It examines the presence of human-on-AI and AI-on-human interactions, offers insights on the use of machine learning to improve game complexity, create adaptive challenges, and personalize player experiences. The result is a testament to the potential of combining foundational game design with state-of-the-art AI, providing lessons for both games of the future and educational applications.**

***Index Terms*—Unity ML-Agents Toolkit, Tetris, Reinforcement Learning, AI-on-human interactions, Adaptive Challenges**

I. Introduction

First released in 1984, Tetris is a simple enough puzzle game made strategic by its deceptively simple play style, it fills an evergreen desire to play with competing a shared available collective memory of the game. The goal—arranging tumbling tetromino blocks to form solid rows—has stressed players' reflexes, spatial understanding and problem-solving prowess for generations. As technology improves for gaming, so does the ability to reimagine something like this using the power of innovative technologies such as artificial intelligence (AI) and machine learning.

This paper reinvents the Tetris experience by combining cutting-edge AI using the Unity ML-Agents Toolkit. The toolkit allows developers to innovate and implement intelligent agents that learn from their surroundings and improve their performance over time. By applying reinforcement learning methods , these agents can analyze the state of the game, predict optimal moves, and adapt their strategies dynamically to compete with or assist human players.

Beyond creating AI opponents, the game incorporates adaptive puzzle game plays where difficulty scales in real time based on player performance. The visual enhancements provide a modern infrastructure while maintaining the core gameplay mechanics that make Tetris engaging. The introduction of rotating blocks adds an extra layer of complexity, requiring players and AI agents alike to think critically about placement and orientation strategies.

This fusion of orthodox gameplay with advanced machine learning serves multiple purposes. From an entertainment perspective, it offers players a fresh take on a classic game. From an educational perspective, it demonstrates the potential of AI in game design, providing insights into how intelligent systems learn, adapt, and enhance user experiences. This project not only pushes the boundaries of interactive entertainment but also serves as a case study in applying AI to develop challenging, adaptive, and innovative games.

II. Literature Survey

[1] Aviv Elor et al. presents a novel artificial intelligence-powered game mechanism designed to utilize the processes of performing physical exercises within an immersive virtual reality game environment, employing deep reinforcement learning techniques such as l Imitation Generation Learning and Optimization of Effective Policies. The findings indicate that deep learning agents are capable of effectively learning gaming-related exercises, offering users unique insights and enhancing their experience.

[2] Youssef et al. demonstrates that applying imitation learning on top of reinforcement learning significantly reduces the amount of training data required and the computational time.The task is learned by seeing an expert demonstrate it. Every player imprints his behavior on the clone through the use of imitation learning.Because they are imitating the main character, each agent will play differently.

[3] Raut et al. In this work, the fundamental concepts of reinforcement learning are examined, along with the mechanisms that underlie this paradigm. With an emphasis on the gaming industry, they examine the subtleties of Reinforcement Learning algorithms and methodologies, allowing developers to produce non-player characters that present unique and tailored challenges to gamers.

[4] Kim et al. The DQN-based control algorithm taught was trained using three artificial ground movements, and its control efficiency was examined using a second simulated earthquake. For comparative analysis, the passive on scenario with the highest damper force was used. The results demonstrate that the Unity game engine may effectively be used to generate an RL environment for structural dynamic systems by appropriately presenting the dynamic responses of the SBIS.

[5] Urmanov et al. When taken as a whole, these experiments demonstrate that research on the reinforcement learning approach may be used to create a training agent.In most cases, it is preferable to develop a traditional scripted AI

implemented by state machines or behavior trees, even though the outcomes of training machine learning agents depend heavily on a number of factors, including hyperparameter setting, environment rules, reward and penalty design, and training time.

[6] Berta et al. Terminal conditions, environment complexity, and reward function hyperparameters were among the various phases that underwent alterations based on how they changed over the course of several training attempts. For efficient training, it was essential to adjust the simulation tick and decision period parameters. Because high-level notions are abstracted, the agent must be trained in environments that are sufficiently complex in terms of the quantity of obstacles.

[7] Marín-Lora et al. The iconic Tetris game, whose fundamental design implies that its implementation should be made up of vector and matrix data structures, has been utilized as a case study. Validating the use of a multi agent based game specification for the game's cross-platform implementation is the goal.This work establishes the findings can be transferred across platforms, based on a study for the definition of games that permits the production of games as multi-agent systems.

[8] Lora-Ariza et al. The player's Tetris skill level is dynamically assessed using case based reasoning, and game difficulty is adjusted based on the complexity of recent game states. A method inspired by flow theory and CBR was proposed, aiming to replicate difficulty patterns that previously induced flow in players with similar skill levels. techniques are employed to adapt video game difficulty based on player performance. DDA system was developed for Tetris and tested on 40 participants.

[9] Iwai et al. Game-playing AIs have surpassed humans in many areas, with examples like Tetris gaining recognition . Researchers tested six algorithms to evaluate their effectiveness. Algorithms , which predict multiple moves ahead, outperformed traditional methods. These findings align with (DDA) systems, as both aim to optimize gameplay by adapting to in-game states.

[10] Bergdahl et al. Using a user-defined, reinforcing reward signal, Deep Reinforcement Learning allows the model to explore and exploit the game systems. This adds on to the test coverage and leads to the discovery of glitches, weak points , and unnecessary mechanics in a wide variety of game kinds. Deep Reinforcement Learning is used to detect frequent issues that occur during the testing and to expand test details , find targeted weak spots, and mapping complexity.

III. Methodology

In adapting Tetris into Unity, the first step is implementing the core game mechanics, which is primarily handled by the 'Piece.cs' script. This script defines the behavior of the tetrominoes, such as their shapes, movement, and rotation within the game grid. It uses a two-dimensional array to track the positions of all pieces and their interaction with the grid. The script allows user input to control the tetrominoes, such as rotating them or moving them left and right. Additionally, the 'Piece.cs' script handles the "falling" mechanic, which is central to the Tetris gameplay, where pieces automatically drop down after a set interval. This mechanic is managed through a combination of functions that control the speed and positioning of each piece.

The second critical element is line clearing, which is handled in the 'Board.cs' script. This script scans the grid for any completely filled lines and clears them, causing the blocks above to fall down to fill the gap. The clearing of lines is a vital part of the Tetris experience, as it provides a reward system and increases the game's difficulty by creating more space for new pieces. The 'Board.cs' script also interacts with the 'Piece.cs' script to ensure that the movement and rotation of the tetrominoes within the grid are accurate and that the game logic for clearing lines functions smoothly. The clearing process involves checking each row for filled lines, and if a line is complete, it is removed, and the remaining lines are shifted down.

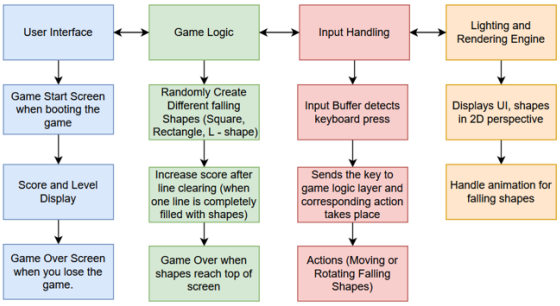

Fig. 1. Architecture of Game

The third script, 'Tetromino.cs', manages the game-over condition. This script monitors the position of the falling pieces and checks if they collide with existing blocks in such a way that they block the placement of new pieces. In Figure 1, If this happens, the game-over condition is triggered, and the game ends. The 'Tetromino.cs' script also resets the game state, clearing the grid and preparing for a new round. This ensures that the player can start a new game immediately after a game over, maintaining an engaging user experience. The game-over logic also includes keeping track of the player's score and updating the UI to reflect the current game state.

Finally, the 'Data.cs' script integrates reinforcement learning to control the AI's behavior. Unlike traditional AI that follows a set pattern or script, reinforcement learning allows the AI to learn optimal strategies over time by interacting with the game environment. The 'Data.cs' script uses a reward and punishment system where the AI is rewarded for clearing lines and penalized for poor moves, such as leaving gaps or stacking blocks too high. The AI learns by playing through numerous game iterations, refining its strategies and improving its performance. This script allows the AI to adapt to various game situations, offering a challenging opponent for the player and demonstrating the potential of machine learning in video games. Through this integration, the AI becomes

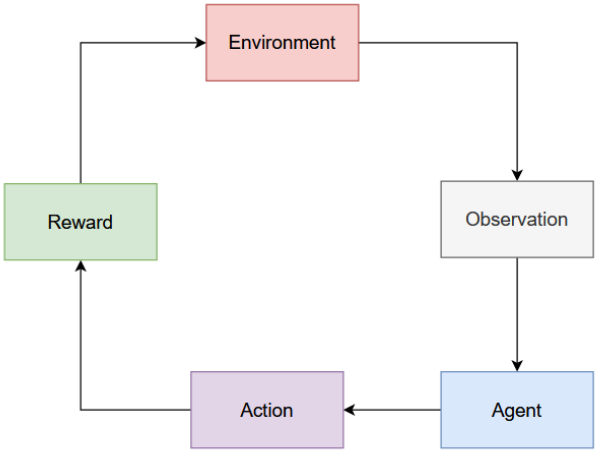

Fig. 2. Reinforcement Learning Training Loop

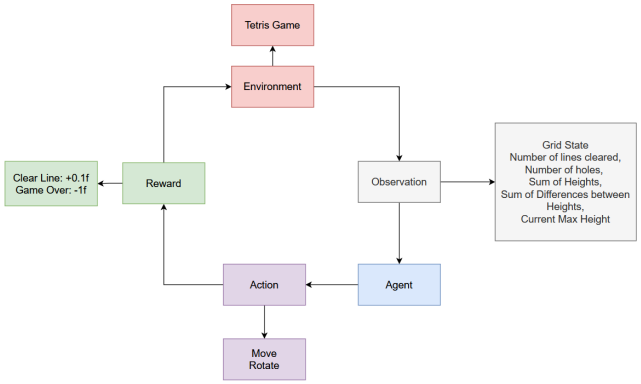

Fig. 3. Architecture of our Model

more dynamic, continuously improving and providing a richer gameplay experience for the player.

We have used reinforcement learning to train an AI to play our game using Unity ML Agents toolkit. First we installed the ml agents package corresponding to our Unity Editor version. If the version of ML Agents differs from the version of Unity Editor, ML Agents package will not work. Python along with numpy, pandas and pytorch libraries also has to be installed. To train a model, first the parameters of the model have to be defined in a tetrisconfig.yaml file. Then we can start training by typing mlagents-learn tetrisconfig.yaml. If we click on play button the Unity Editor will connect with the python program and training will start.

In Fig. 2, the reinforcement learning training loop is shown. Here, the Environment is the tetris game, Observations are the states of the game that is position of the piece, placement of blocks, score and number of lines cleared, etc. Agent will take these observations as input and output an action into the game, for example, moving a piece. Based on the action, the agent will receive a reward or penalty. Rewards can be given for clearing more lines, while penalty is given for game overs.

Fig. 3 shows the architecture for our model. The Environment is the Tetris Game. We have specified the following parameters for observations. There are a total of 215 observations for our game that is input to the agent. The x and y coordinate of the piece that is currently falling down, The rotation angle of the piece, the coordinates of borders of the game board, the current board state - that is how many tiles are filled, how many are empty which will help agent decide where to place the falling piece to get higher scores. We also added number of holes, number of lines cleared, the current score, sum of heights, sum of differences between heights. To input these observations to the agent, we ovveride the CollectObservations method in the Unity.MLAgents library.

To give player control to the RL agent, we override OnActionsReceived() method and specify what objects the agent can control. We have given the same actions as the player, for example move left, right, down and rotate clockwise or anticlockwise. This method also calls a Reward() method that will give a reward to the model based on its actions.

In the reward() method, to prioritize higher score, we give rewards to the agent if the agent makes actions that clear more lines. We also add penalty if the agent increases the height of the columns formed by pieces falling, since the game will end if the pieces reach the top of the board. If the agent reaches game over condition, it is added heavy penalty. Higher rewards are given for higher scores.

IV. Results and Analysis

We have used Proximal Policy Optimization algorithm to train the reinforcement learning model. To train the model, we have considered the following parameters: Buffer size of 4096 specifies how many previous experiences the model can store in main memory. Learning rate of 3e-4 specifies how the model's gradients will be updated. The input features are normalized. The model can use multithreading to speed up the training process. We have specified value of gamma as 0.99 which indicates that future rewards have more priority than current rewards. A step is one loop through the Environment, Observation, Agent, Action, Reward cycle. The max steps are 4e5. After training is done, the model is stored in the results folder. The model can be loaded and used for inference later.

Fig.4 shows the Policy Loss Graph. Policy Loss is a performance metric in reinforcement that measures how well the policy function is able to map the states into actions. That is given a state in the game, the policy function will decide which action to take to get greater rewards or higher score. The y axis shows loss in policy, and x axis shows number of steps taken. It indicates that over time, after large number of steps the policy loss is minimized.

Fig.5 shows the cumulative reward graph. Cumulative reward is the sum of rewards accumulated over episodes. An episode starts when the game starts and ends with game over condition. The score as well as the rewards are stored with each episode. As seen in the graph as the number of steps are increased, the reward is also increased.

Fig. 6 shows a screenshot of the implemented game, along with levels, score and high score. After training, the model

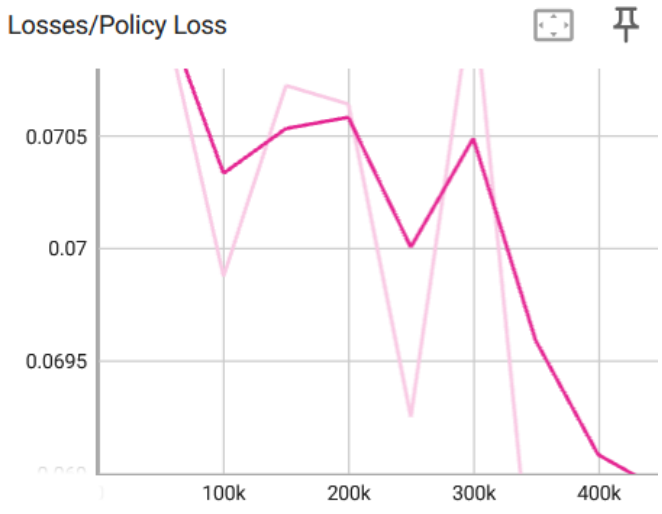

Fig. 4. Policy Loss Graph

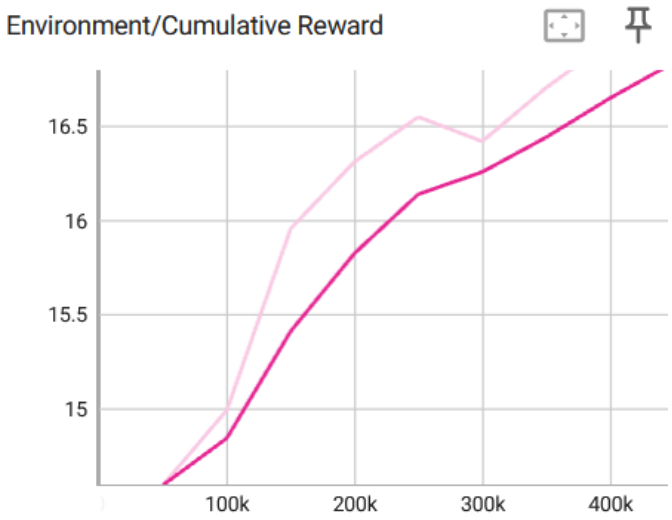

Fig. 5. Cumulative Reward Graph

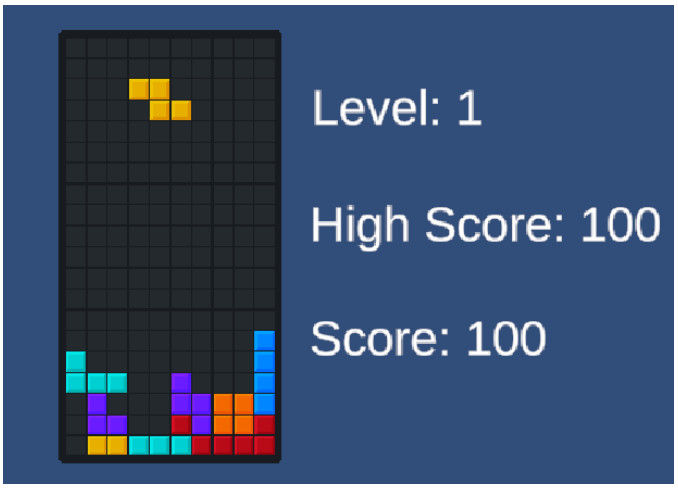

Fig. 6. Screenshot of Tetris Game

was able to achieve a high score of around 1200. The hardware that this model was trained on does not have a GPU and hence limits the performance of the model. We also tried the model on a variant of tetris with only square boxes. Here as there is only piece to control, the model achieved a higher score of 3000.

V. Conclusion and Future Work

The integration of core Tetris mechanics in Unity, combined with reinforcement learning, creates a dynamic and engaging gameplay experience. The scripts—Piece.cs, Board.cs, Tetromino.cs, and Data.cs—work together to ensure smooth gameplay, line clearing, game-over handling, and intelligent AI. The use of reinforcement learning enhances the AI's adaptability, making the game more challenging and rewarding.

For future work, further optimization of the AI's learning process could be explored, potentially integrating advanced neural networks. Additionally, expanding the game features with multiplayer capabilities and enhanced graphics would enrich the overall player experience. The model could be trained on better hardware to improve performance of the model.

References

[1] Elor, A., Kurniawan, S. (2020, August). Deep reinforcement learning in immersive virtual reality exergame for agent movement guidance. In 2020 IEEE 8th International Conference on Serious Games and Applications for Health (SeGAH) (pp. 1-7). IEEE.

[2] Youssef, A. E., El Missiry, S., El-gaafary, I. N., ElMosalami, J. S., Awad, K. M., Yasser, K. (2019, October). Building your kingdom imitation learning for a custom gameplay using unity ml-agents. In 2019 IEEE 10th Annual Information Technology, Electronics and Mobile Communication Conference (IEMCON) (pp. 0509-0514). IEEE.

[3] Raut, U., Galchhaniya, P., Nehete, A., Shinde, R., Bhoite, A. (2024, July). Unity ML-Agents: Revolutionizing Gaming Through Reinforcement Learning. In 2024 2nd World Conference on Communication Computing (WCONF) (pp. 1-7). IEEE.

[4] Kim, H. S., Kang, J. W. (2024). RL-based Control of Smart Base Isolation System Using Unity ML-Agents. International Journal of Steel Structures, 24(4), 908-917.

[5] Urmanov, M., Alimanova, M., Nurkey, A. (2019, December). Training unity machine learning agents using reinforcement learning method. In 2019 15th International Conference on Electronics, Computer and Computation (ICECCO) (pp. 1-4). IEEE.

[6] Berta, R., Lazzaroni, L., Capello, A., Cossu, M., Forneris, L., Pighetti, A., Bellotti, F. (2024). Development of deep-learning-based autonomous agents for low-speed maneuvering in Unity. Journal of Intelligent and Connected Vehicles, 7(3), 229-244.

[7] Marín-Lora, C., Chover, M., Sotoca, J. M. (2022). A multi-agent specification for the Tetris game. In Distributed

Computing and Artificial Intelligence, Volume 1: 18th International Conference 18 (pp. 169-178). Springer International Publishing.

[8]Lora-Ariza, D. S., Sánchez-Ruiz, A. A., González-Calero, P. A., Camps-Ortueta, I. (2022). Measuring control to dynamically induce flow in Tetris. IEEE Transactions on Games, 14(4), 579-588.

[9] Iwai, H., Sato, K., Alparslan, O. (2024, August). Comparison of Game Playing AI Algorithms in Puyo Puyo Tetris. In 2024 IEEE Conference on Games (CoG) (pp. 1-2). IEEE.

[10] Bergdahl, J., Gordillo, C., Tollmar, K., Gisslén, L. (2020, August). Augmenting automated game testing with deep reinforcement learning. In 2020 IEEE Conference on Games (CoG) (pp. 600-603). IEEE.