RAINBOW DQN APPLIED TO IZOMBIE A STRATEGIC PUZZLE GAME

Daniel Chen, William Hu, Yunhan Mao, Mengfei Qi & Shuning Zhang
Department of Computer Science
University of Southern California
Los Angeles, CA 90007, USA
{dchen110,huwillia,yunhanm,mengfeiq,shuningz}@usc.edu

Code of this project: `https://github.com/Lily-Qi/IZombie` Please follow the `README` instructions if you want to replicate the result.

1 INTRODUCTION

Plants vs. Zombies (PvZ) is a popular strategy game where players strategically place plants with various defensive abilities to stop waves of zombies from reaching their house. While many programs have tried to train models to play the traditional PvZ, our project explores advanced machine learning models, specifically Deep Q-Network (DQN) integrated with Multi-Layer Perceptron (MLP) and Long Short-Term Memory (LSTM), to master "I, Zombie" mode, where players spend suns–the in-game currency–to "plant" the zombies and destroy plants while trying to eat all 5 brains at the end of each row. Winning in this mode is defined by successfully navigating zombies through the plant defenses to eat all 5 brains, and losing occurs when players fail to consume all brains when they run out of suns to deploy additional zombies. Therefore, a balance between the attacking strategy and sun conservation is critical for success. Our goal is to train an AI agent not only to win but also to maximize the remaining suns when winning the game. Before our project, there has been no learning models developed looking from the adversarial point of view of the zombies. Mastering this unique viewpoint offers significant insights into strategic decision-making and efficient resource management. This project may have potential applications in future game development, particularly in terms of game theory. It could help formulate adversary defense strategies or develop sophisticated AI for games in general. Furthermore, the ways to develop strategic resource allocation skills may also be extended to other areas that require similar strategic insight and resource management capabilities.

Many existing AI models for PvZ have primarily focused on the traditional game mode, neglecting the unique strategic elements of the "I, Zombie" mode. More particularly, the models for the traditional PvZ lacked the capability to strategize for more effective sun management. In the traditional gameplay, the in-game currency, suns, can be generated by sun-producing plants, such as Sunflower or Sun-shroom, and dropped from the sky in the daytime levels, which means there is no upper limit for the suns that players can collect. However, in the "I, Zombie" mode, suns can only be obtained by destroying Sunflowers, which implies that the maximum number of suns players can collect is determined when the game starts. Therefore, effectively managing the resources is even more critical in this reverse role-play scenario. To better define the problem, the action spaces for the game is discrete, considering that choosing which type of zombies can be placed in a limited number of grids. However, the state of the game can be really complex, involving the number of remaining suns as well as various plants and zombies with their types, locations, and health points (HP). Deep Q-Network (DQN) is then selected to solve this task considering its effectiveness in discrete action spaces and complex environments. It is significant to notice that DQN used to play games where the reward is not delayed. However, our project tries to teach agents the importance of long-term strategic planning and decision-making in different ways, such as integrating DQN with Long Short-Term Memory (LSTM) models, so that the agents can realize that their actions have long-term consequences. This ability for long-term planning is important for efficient sun management in the game.

Our project's contributions are multifaceted. We customized an open-source PvZ simulator for the "I, Zombie" mode, facilitating efficient data collection and agent training. Beginning with a ba-

sic DQN model, we sequentially incorporated Double DQN (DDQN), Rainbow DQN, and finally LSTM to optimize strategic long-term planning and sun management. Through comprehensive performance analysis, focusing on metrics like winning rate and average remaining suns, we gained insights into each model's ability to balance immediate objectives with long-term resource conservation.

2 Related Work

Through the inspiration of the work of (Timothée,, 2021), which presented three different agents, actor critics, Deep Q-Network (DQN) and Double Deep Q-Network (DDQN), to train them to play the classic mode of *Plants vs. Zombies* (PvZ), our team decided to train two different agents with improvements in their replay buffer and architectures and compare their performance in terms of win-rates and remaining suns after winning. In terms of the choice of the linearity of the ϵ-greedy strategy, our team inherited the choice presented in (Timothée,, 2021) to use exponential ϵ-greedy strategy. The benefits of the choice are also confirmed in (Rodrigues, and Vieira, 2020) that prioritized random exploration at the beginning of the training can better avoid the problem of overfitting. Furthermore, our team also built a customized simulator for the "I, Zombie" mode of the game and feeds input states of the game per frame as the methodology to update information to the agents. However, we improved upon their work by using one of the state-of-the-art agents in DQN, Rainbow DQN (Hessel,, 2018), and incorporated recurrence by adding Long Short-Term Memory (LSTM) and Upper-Confidence Bounds (UCB) in hopes of boosting its performance by letting agents relate to longer sequences of actions. "I, Zombie" mode of PvZ is a classic tower defense game, which aligns with the Markov Decision Problem (MDP). It has been proven that DQN and DDQN are effective strategies for MDP in (Souchleris,, 2023). Also, as a fast decision game that requires the player to make real-time decisions while the game runs, it is stated by (Liang, and Li, 2022) that the most effective way is to use DDQN.

The implementation of DQN was first introduced by a team from DeepMind (Mnih,, 2013), which prior to the publication of the article Q-Learning was the main go-to method for various tasks. The DeepMind team was the first to integrate Deep Reinforcement Learning with Q-Learning and proved that the application in complicated games with DQN is possible, as the team approached 7 different famous Atari games with DQN, and 4 of them had super-human performance. So much more has now been improved from the basic agent the team developed years ago, including the development of Rainbow DQN and DDQN. The choice of using Deep Reinforcement Learning as our main approach is confirmed by (Oroojlooyjadid,, 2021), which states that it is best to train with a DQN agent if our problem observes the full state variables and optimizes a single agent. An improvement we have made upon this notion is that we further used DDQN to avoid overestimation of the current training data. In addition, our team also implemented Prioritized Experience Replay (PER) instead of the traditional uniform experience replay in a standard DQN agent for game applications (Tom,, 2015). PER allows random selection of previous experiences depending on the magnitude of the Temporal Difference (TD) error. This method leads to a quicker convergence in the remaining sun after around 1.8 million episodes since it would stabilize the training and cause convergence in the loss (Bakhanova, and Makarov, 2021). Furthermore, the training of DQN agents often directly translates pixels of the game state through Convolutional Neural Network (CNN) to the agents (Paudel,, 2020). As an improvement, we have decided to customize a simulator that runs in C++ and provides the states in an array form to increase training speed and efficiency. The inspiration for incorporating LSTM in Rainbow DQN and DDQN is from (Moreno-Vera,, 2019), which they implemented LSTM to DQN and DDQN. The incorporation of LSTM in Rainbow DQN that boosts the win-rate to have super-human performance is an improvement upon their work.

In regards to work related to future work, such as considerations of upgrading the agents under the Partially Observed Markov Decision Problem (POMDP) (Dmitry, and Makarov, 2019), Monte Carlos Tree Search (MCTS) (Zhang,, 2020), and Deep Spiking Q-Network (DSQN) (Liu,, 2023), they are discussed in greater detail in the conclusion and future work section.

3 PROBLEM FORMULATION

3.1 THE GAME: I,ZOMBIE MODE IN PLANTS VS. ZOMBIES

Our project focuses on the I,Zombie mode, a puzzle game in "Plants vs. Zombies." We utilized an open-source simulator[1] that replicates all the logics of plants and zombies in the game. To facilitate our study, we modified this simulator to include the IZombie mode, with a specific enhancement: the sun count increases whenever a sunflower is eaten, and also to return the state variables that we needed.[2] This simulator allows for direct acquisition of state data post-action, eliminating the need for visual state analysis and enabling more efficient agent training.

The game board consists of 5 rows and 9 columns. The left 4 columns are initially populated with 9 sunflowers, 6 peashooters, 3 squashes, and 2 snow peashooters, arranged randomly in each episode. The player starts with 150 suns and can deploy zombies only in the right 5 columns. There are three zombie types available: normal zombie (costing 50 suns), buckethead zombie (125 suns), and football zombie (175 suns).

3.2 STATE REPRESENTATION

Our initial state input included the types and HPs of plants in the left four columns, types and summed HPs of zombies in each grid (including grids without zombies), the number of suns, and the presence of brains in each row. This initial state comprised 136 entries: 4x5 for plant types, 4x5 for plant HPs, 9x5 for zombie types, 9x5 for sum of zombie HPs, 1 for sun count, and 5 for brain presence.

Realizing this was inadequate for comprehensive game understanding, we expanded the state. Given the maximum of 1950 suns and up to 39 zombies, we now account for each zombie's type, HP, row, x-axis position, accessory HP, and slow motion countdown (due to snow peashooters). Plant data now includes type, HP, row, and column. Additionally, we track current and spent suns, and brain presence in each row. This results in a total of 321 entries: 39x6 for zombies, 20x4 for plants, 1 for current suns, 1 for spent suns, and 5 for brain presence.

3.3 ACTION SPACE

Initially, we provided a small action space so that we could find a suitable algorithm faster. The action space was limited to allow the agent to place any of the three zombies only in the 5th column, and not in rows without a brain. This resulted in a 16-action space (5x3 for each zombie type in the 5th column, plus 1 for no action). After identifying a suitable algorithm, we removed all constraints. So, we expanded the action space to include all 5 columns, resulting in a 76-action space (5x5x3 for each zombie type in each column, plus 1 for no action).

3.4 REWARDS SYSTEM

With the initial limited action space, the reward was based on the difference in suns between states, normalized by 25. This encouraged efficient sun use to win the game since the number of suns will be 0 if the agent loses the game. After expanding the action space, we introduced a more complex reward system based on the previous one: 2 points for each new plant eaten, 8 points for each brain consumed, and a 5-point penalty for losing. This aims to teach the agent to place zombies effectively, target plants strategically, and avoid wasting suns on empty lanes.

3.5 EVALUATION OF AGENT PERFORMANCE

Our evaluation focuses on two key metrics: the winning rate and the average remaining suns in victorious episodes. We calculated the winning rate by having the agent play 500 episodes, then averaging the rate of successful games. This metric assesses the agent's ability to achieve the game's primary goal. Additionally, we measured the average remaining suns in winning episodes to evaluate

[1]Original simulator: `https://github.com/dnartz/PvZ-Emulator`

[2]Modiefied simulator: `https://github.com/Rottenham/PvZ-Emulator`

resource efficiency. This dual metric approach ensures the agent is not only successful in winning but also proficient in managing resources, a vital aspect of strategic gameplay in I,Zombie.

4 METHODOLOGY

We initiated our research by modifying an existing open-source "Plants vs. Zombies" simulator, specifically tailored for the "I, Zombie" mini-game. This modification was crucial as it enabled the direct acquisition of state data following each action, thereby eliminating the traditional need for visual state analysis. This approach significantly improved our data collection process, providing a more efficient way for our subsequent agent training tasks.

Our primary model was a Deep Q-Network integrated with Multilayer Perceptron. Chosen for its effectiveness in goal-specific environments, the DQN framework excels in making decisions based on the current state. However, one of the limitation of DQN is its tendency to overestimate Q-values, as highlighted by (Van,, 2016), which was a essential factor in the evaluation and fine-tuning of our model's performance.

To counteract the overestimation in traditional DQN model, our methodology progressed to incorporate Double DQN. DDQN updates the Q-values using two separate networks to mitigate the overestimation of Q-values. It will use one of the networks to choose the best action and another to estimate the q-value of that action. This approach was essential in refining the model's performance, based on the findings of (Van,, 2016). The formula for the DDQN update rule is as follows *:

$$Q_{update} = r + \gamma \cdot Q_{target}\Big(s', \arg\max_{a'} Q_{current}(s', a')\Big) \tag{*}$$

In this formula, Q_{update} denotes the updated Q-value for being in state s and taking action a. r is the reward received after taking action a in state s. γ is the discount factor which determines the importance of future rewards. It is a value between 0 and 1, where a value closer to 0 makes the agent focus more on the actions which can bring immediate rewards, and a value closer to 1 leads to a better focus for future rewards. $Q_{\text{target}}(s', a')$ is the Q-value predicted by the target network for the next state s' and the action a' that maximizes the current network's Q-value for the next state. This is where DDQN differs from standard Q-Learning. $\arg\max_{a'} Q_{current}(s', a')$ is the action that maximizes the Q-value for the next state s' according to the current Q-network, which is used to select the action a'.

Then, we continued our research and decided to move on to RainbowDQN, which is another varient of DQN that has the best performance when it comes to handling the unpredictability of starting conditions and learning complex strategies. The name Rainbow comes from the various components it encompasses. The model include these following features besides DDQN:

- **Prioritized Replay** adjusts the agent's learning focus by replaying important experiences more frequently, thereby improving learning efficiency.
- **Dueling Networks** feature an architecture that separately estimates the state value and the advantages of each action, providing a more precise assessment of the value of each action.
- **Multi-step Learning** extends the lookahead of the agent by considering a sequence of actions, which helps in understanding the consequences of actions over a longer horizon.
- **Noisy Nets** introduce stochasticity into the network parameters to aid in exploration, reducing the need for an external exploration strategy like epsilon-greedy.

These components collectively enhance the long-term strategic learning process, as depicted in Figure 3 and as suggested by (Hessel,, 2018).

We also explored the integration of Long Short-Term Memory networks, supplanting the initial MLP component. This replacement aimed to leverage this temporal dynamic, hypothesizing that it would enable the agent to more effectively learn policies that depend on long-term planning and strategy. This is particularly pertinent in "Plants vs. Zombies," where an agent must plan several moves ahead and consider the long-term implications of current actions, such as resource (sun) accumulation. By integrating LSTM networks, we anticipated a more sophisticated model capable of capturing these temporal relationships, leading to improved performance in our simulations.

5 RESULTS

Since the main goal of our agent is to win the game with as much remaining sun as possible, the aforementioned different models were accessed based on the metrics of winning rate and average remaining sun as the indicators of their performance. The obtained results are visually represented in the graphs below.

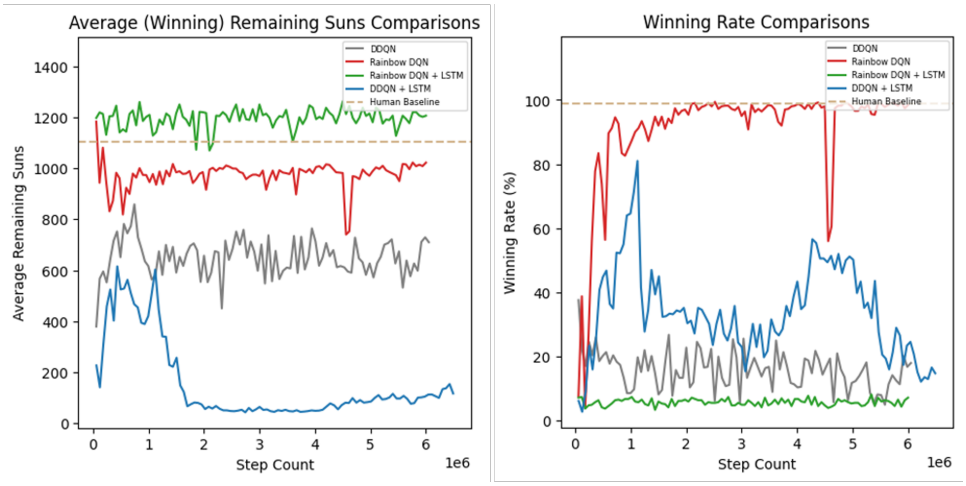

Figure 1: Performance of different agents.

The DDQN with a MLP architecture exhibited lower winning rates and average remaining suns throughout the training process, as indicated by the gray line in the graphs. These findings suggest that while DQN and DDQN could navigate the game environment to a certain extent, their ability to optimize gameplay in a manner that maximizes the long-term rewards was limited.

However, the implementation of Rainbow DQN marked a significant improvement in both metrics. The red line in the graphs illustrates a higher level of average remaining suns and a much higher winning rate. This substantial enhancement in performance underlines the capability of Rainbow DQN to effectively manage the game's strategic complexity, balancing immediate actions with their long-term outcomes more proficiently.

Adding to this, the DDQN+LSTM model (depicted with the blue line) achieved a better winning rate than DDQN, but it also showed a lower level of average remaining suns than DDQN. On the other hand, the integration of LSTM with Rainbow DQN, intended to further enhance decision-making through an understanding of temporal sequences, maintained the highest average of remaining suns but paradoxically resulted in the lowest winning rates among the tested models, as seen in the green line of the winning rate graph. These unexpected outcomes suggest that the reward function could potentially be further optimized for the LSTM's pattern recognition capabilities. In consideration of our human baseline for remaining suns that has a mean of 1104.76 and a standard deviation of 107.77—which suggests the human baseline is a stable dataset—the integration of LSTM with Rainbow DQN has surpassed the mean baseline 100% of the time, depicting it having learned the most reliable and effective strategies in comparison to all the other models and average human performances.

Since Rainbow DQN had the best overall performance, we attempted to fine-tune some of the training parameters to further optimize its performance, as shown in the following figures.

First, we can see from the figures that when n, i.e. the number of steps used in the multi-step learning part of Rainbow DQN, is kept at 3, adjusting the batch size from 128 to 256 boosted the model's performance, in terms of both winning rate and average winning sun, by a slight margin.

In addition, when fixing the batch size at 256, changing the number of steps from 3 to 10 demonstrated even better performance. However, it should be noted that both the above figures were obtained when constraints were removed. Specifically, the agent is no longer banned from plac-

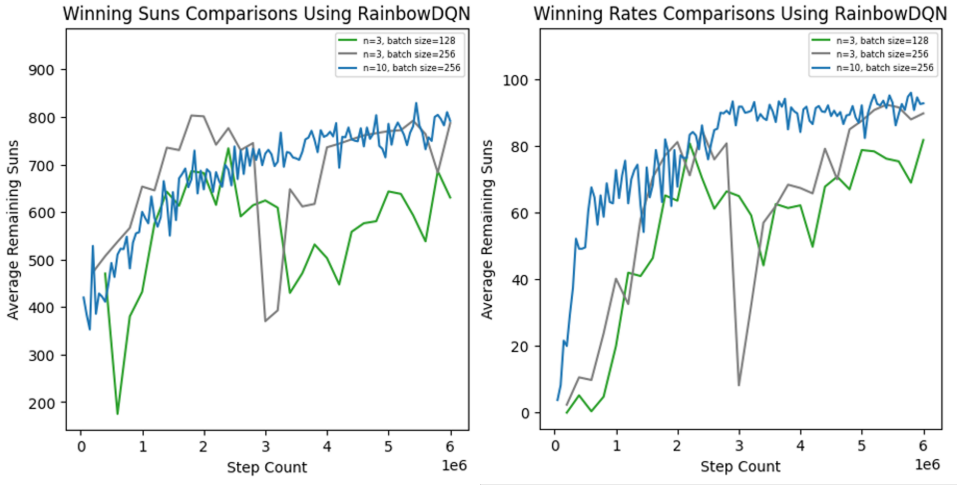

Figure 2: Performance of Rainbow DQN agents trained with different parameters.

ing zombies at column 6 to 9 or at rows with brain already eaten. Hence, while the winning rate approaches 100% and matches the human baseline, the average winning sun is considerably lower than Rainbow DQN with constraints enabled. This was done as part of the further exploration to see how well agents could perform without manually set constraints on action space.

6 Conclusion and Future Work

This research demonstrates the differential impact of neural network architectures on AI performance in the "I, Zombie" minigame. The Rainbow DQN with MLP achieved the highest winning rates, highlighting the strength of advanced DQN modifications for decision-making tasks. Conversely, the Rainbow DQN + LSTM, while showing lower victory rates, maintained a high level of remaining suns, pointing to its potential for creating efficient resource management strategies. The significance of these findings lies in their illustration of the complex trade-offs between achieving immediate objectives and maintaining resource efficiency. The results guide us in choosing the right AI model based on the desired outcome, whether it's maximising success or conserving resources. Future work will focus on refining the understanding of when and how LSTM can be leveraged to improve performance, possibly by adjusting the replay buffer of the learning algorithm to adapt LSTM better, to yield a model with both high winning rates and suns remaining.

Future work could also be looked into incorporating POMDP with our current Rainbow DQN with LSTM and DQN with LSTM methods. A specific training scenario could be that zombies only see the first 4 rows of plants, with more rows off-screen to the left that shows as the game progresses (Dmitry, and Makarov, 2019). Another idea is to add Spiking Neural-Network (SNN) to DQN, creating DSQN models for training. This new method has shown better performance than traditional DQN in most of the top 17 Atari games (Liu,, 2023). Alternatively, using MCTS as a training algorithm could be explored, as the game involves long-term, deep strategic planning with minimal randomness and a finite state space for each episode, which MCTS handles well (Zhang,, 2020). Further research could also see if these findings apply to other strategic games or real-world situations like resource allocation and logistics, to see how widely these AI training methods can be used.

References

Bakhanova, Maria and Ilya Makarov. Deep reinforcement learning in vizdoom via dqn and actor-critic agents. In *Advances in Computational Intelligence: 16th International Work-Conference on Artificial Neural Networks, IWANN 2021, Virtual Event, June 16–18, 2021, Proceedings, Part I 16*. Springer International Publishing, June 2021.

Dmitry, Akimov and Ilya Makarov. Deep reinforcement learning with vizdoomfirst-person shooter. *Munich Personal RePEc Archive*, pp. 3–17, Semptember 2019.

Hessel, Matteo, et al. Rainbow: Combining improvements in deep reinforcement learning. In *Proceedings of the AAAI conference on artificial intelligence*, volume 32.1, November 2018.

Liang, Hai and Jiaqi Li. A study on the agent in fighting games based on deep reinforcement learning. *Mobile Information Systems*, July 2022.

Liu, Guisong, et al. Human-level control through directly trained deep spiking q-networks. In *IEEE Transactions on Cybernetics*. IEEE, April 2023.

Mnih, Volodymyr, et al. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, December 2013.

Moreno-Vera, Felipe. Performing deep recurrent double q-learning for atari games. In *2019 IEEE Latin American Conference on Computational Intelligence (LA-CCI)*. IEEE, November 2019.

Oroojlooyjadid, Afshin, et al. A deep q-network for the beer game: Deep reinforcement learning for inventory optimization. *Manufacturing & Service Operations Management*, 24.1:285–304, February 2021.

Paudel, Prabesh. Exploring game playing ai using reinforcement learning techniques. Novemeber 2020.

Rodrigues, Pedro and Susana Vieira. Optimizing agent training with deep q-learning on a self-driving reinforcement learning environment. In *2020 IEEE Symposium Series on Computational Intelligence (SSCI)*. IEEE, December 2020.

Souchleris, Konstantinos, et al. Reinforcement learning in game industry—review, prospects and challenges. *Applied Sciences*, 13.4, February 2023.

Timothée, Blondiaux, et al. Plants vs. zombies: Reinforcement learning to a tower defense game. 2021.

Tom, Schaul, et al. Prioritized experience replay. *arXiv preprint arXiv:1511.05952*, 2015.

Van, Hasselt, et al. Deep reinforcement learning with double q-learning. In *Proceedings of the AAAI conference on artificial intelligence*, volume 30.1, 2016.

Zhang, Amy. Monte carlo tree search applied to plants vs. zombies. 2020.

A Appendix

A.1 Work Distribution

Daniel Chen: Modified the PVZ simulator and environment, implemented DDQN+MLP and RainbowDQN+MLP training, tested the effectiveness of different parameters on training, and wrote the result part of the report.

William Hu: Implemented environment including the state parsing, tested the effectiveness of different parameters on training, did research on related works, wrote the related work and conclusion part.

Yunhan Mao: Proposed the use of RainbowDQN and LSTM, implemented LSTM in DDQN and RainbowDQN to adapt the sequential feature, wrote the method part.

Mengfei Qi: Implemented environment including the action space and initial state, tested on different parameters and implemented suitable reward policy for efficient training after removing all constraints, wrote the problem formulation part.

Shuning Zhang: Tested the effectiveness of different parameters on training, got the training result of each model and drew comparison plots, researched suitable reward policy for efficient training after removing all constraints, wrote the introduction part.

A.2 FIGURE

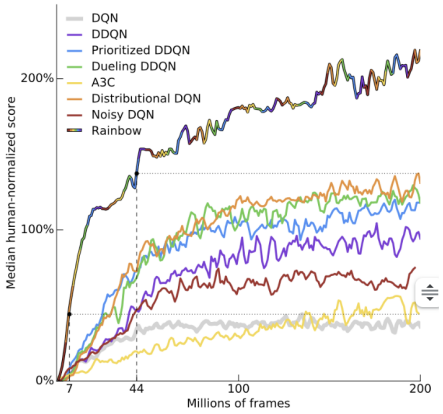

Figure 3: Performance of Rainbow DQN as compared to other DQN models over millions of frames. The median human-normalized score is a metric that normalizes the agent's performance to the human average, where Rainbow DQN outperforms all other considered models.