Alon Biner, 318418118
Mariia Makarenko, 342849676
Yonatan Vologdin, 323316828

Project Proposal

Problem: The goal is to develop an AI player that excels at Jenga, evaluated by specific metrics. In Jenga, players alternately remove one block from a tower made up of 18 layers, each consisting of 3 blocks oriented in alternating directions. The game ends when a player causes the tower to collapse on their turn. We may modify these game parameters to test the AI's adaptability. We believe this problem to be interesting because the AI player needs not just to safely remove upper blocks trying to minimize their own chances of losing, but to strategically choose blocks that make the tower more likely to collapse for the next player, increasing their chances of losing.

Solution: We will build a simulated environment with interface developed in Unity, in which our AI player will learn and play against another AI opponent or a human. We aim for our AI player to achieve good results in this simulated environment, and we don't aim for it to be good in the real world. In this simulation we will model the tower using Unity physics engine. The Unity physics engine will decide whether the tower has to fall, and we will be able to see the current state of the tower through an interface between python code and Unity. The reason for the choice to solve Jenga in a simulated environment is mainly because in the simulation we can remove the factor of simple inaccuracy in pulling the blocks out, which is independent of how good the player's strategy is and can introduce noise in the results. However, we will anyway check if our AI player is good in the real world by reenacting the games from the simulation and comparing the win rates, as an additional evaluation metric.

To solve the problem of selecting which Jenga block to pull without causing the tower to collapse, we will use two approaches: Adversarial Search with Monte-Carlo Tree Search (MCTS) and Reinforcement Learning with Deep Q Network. Monte-Carlo Tree Search (MCTS) is chosen because it effectively models the adversarial nature of Jenga, where each player aims to minimize their likelihood of collapsing the tower while increasing the chances for their opponent. MCTS runs multiple simulations to account for the game's stochastic nature, making it more suitable than deterministic algorithms like Minimax. Model-free Q-learning is selected to enable the AI to learn optimal strategies through trial and error. Given the discrete action space in Jenga (finite number of blocks), Q-learning can efficiently learn from the environment feedback. We opted for the model-free version due to the slow running time of learning Jenga's model. We chose specifically Deep Q Network because of the large number of possible states in hope that a neural network could learn an approximation of the problem with a lower number of states.

Previous work: The only previous work we found is the robot Jenga player built by a team of researchers from MIT (N. Fazeli, M. Oller, J. Wu, J. B. Tenenbaum, and A. Rodriguez) and published in the Science robotics journal and uploaded on Youtube. Their work is significantly different from what we are trying to achieve because they aimed to solve Jenga in the real world using robotics, and we, on the contrary, aim to build an agent that fares well in the simulation. Also, by solving Jenga in the real world they had to use fewer training samples, and so possibly our agent will be better since we can provide it with more training samples in the simulation.

Evaluation: For both Monte-Carlo Tree Search and Deep Q Learning, we will use the following baselines: random move selection and human players using a Unity interface. If we have time we will also do optimistic selection (removing blocks from lower levels based on an adjusted stability score), pessimistic selection (removing blocks from upper levels based on an adjusted stability score). The metrics for evaluation will include the number of moves until the tower collapses, win rate (percentage of games where the opponent collapses the tower), win rate in real-world reenactments, and the sum of all move calculation durations, assuming the opponent takes no time for their moves.

Extra: In the Reinforcement Learning solution, our player will first learn a simplified version of the game (for example, 3 levels) in self-learning. Then we will create a copy of this trained player, such that one player will continue to learn by playing with the other one who stopped learning. We will from time to time swap the player who doesn't learn with the newer version of the player who learns . We will iteratively increase the complexity of the game, and we will consider randomizing the setup, so that the tower would have missing blocks from the beginning. Finally, we will test our trained player by playing with a human.