­­­**Introduction**

In our project we are solving the problem of developing an AI agent that performs well in the game of Jenga. For those not familiar with the rules, the game starts with 54 wooden rectangular blocks (although there are variations of the game with less blocks) stacked in a tower. At each level of the tower there are three blocks placed side by side along their longer side, and the levels alternate direction relative to each other. For example, if in the downmost level the blocks are oriented horizontally by their longer sides, in the next level the blocks should be oriented vertically. The players make moves one after the other, so that in each move a player removes one block from any layer, except the top three, and places it on top of the tower. The player who causes any of the blocks to fall loses the game.

The more blocks are removed from the tower, especially from the lower levels and from the sides, the more unstable the tower becomes and the more likely it is to fall. To succeed in the game, a player needs to make a move that is cautious enough for the tower not to fall in their move, but also risky enough for the tower to become significantly more unstable in the adversary’s move. Due to this tradeoff, there is no trivial strategy the player can adhere to in order to win, and this makes the problem interesting.

As can be inferred already from the description of the game, Jenga is a physical game. In the real-world Jenga game, noise is introduced to the players’ strategies due to imperfect tower stacking, manufacturing tolerances, and friction between blocks. That is, no matter how good their strategies are, the tower can fall if they accidentally touch other blocks when pulling out the block they want to. Because of this, normally players can “probe” the blocks to find the ones that are loose enough to be safely pulled out. Therefore, to create a solver for the real-world Jenga, we would need to use robotics. Since this is not a project in robotics, we decided to simulate the game in its simplified form.

For simulation of the Jenga tower, we turned to Unity and its physics engine. It allowed us to control the parameters of static and dynamic friction, as well as gravity. It also allowed us to know the positions of all the blocks at any given time. First and foremost, it allowed us to isolate the strategic component of the Jenga game by eliminating the noise when pulling out blocks. That is, we were able to implement the process of pulling out of the blocks as if they are pulled without any friction with the neighboring blocks. Furthermore, it allowed our agents to get understanding of the current stability of the tower based on the blocks’ angles and, for example, make more cautious moves when the tower is less stable.

**Previous Work**

In our search for previous work that tried to make an automated solver for Jenga, we managed to find only one paper on the matter, which is a project by a team of researchers from MIT (N. Fazeli, M. Oller, J. Wu, B. Tenenbaum, and A. Rodriguez) published in the [Science Robotics](https://www.science.org/doi/10.1126/scirobotics.aav3123?utm_source=the+new+stack&utm_medium=referral&utm_content=inline-mention&utm_campaign=tns+platform) journal, and with a short video available on [Youtube](https://www.youtube.com/watch?v=o1j_amoldMs) about it. Their project aimed to create a solver for Jenga in the real-world setting and described creating a robot that is capable of seeing and touching. Similarly to the “probing” technique to identify loose enough blocks used by the human players that was discussed earlier, the robot uses tactile feedback to locate the blocks that are loose enough to be safely removed.

Their project, unlike ours, aimed to create a solver for Jenga in the real-world setting, and they created a robot for this purpose. Similarly to the “probing” technique to identify loose enough blocks used by the human players that was discussed earlier, the robot uses tactile feedback as an input for its machine learning algorithm. However, although we hope that our Jenga solver would perform well also in the real-world setting, we aimed for it to solve the simulated version of the game.