Milestone 4 Final Report

CSE 250A / 150A Final Project

Nikitha Maderamitla Arul Mathur Kai Wang Karim Barajas

1 Problem Description

Chopsticks is a two-player hand game involving addition-based attacks and redistribution of finger counts. Each player begins with two hands, each with a single finger held out, denoting a hand value of 1.

On each turn, a player may:

- **Attack**: Add the value of one of their hands to an opponent's hand. Hand values reaching 5 or more become 0 (dead).
- **Swap**: Redistribute their total fingers across their two hands, preserving the sum. Identity and mirror swaps are disallowed. Swaps may revive a dead hand.

A player loses when both of their hands are dead. Despite simple rules, the state space is surprisingly large and strategic. Our objective is to investigate whether **tabular Q-learning** can learn a strong Chopsticks strategy through repeated self-play. We later aim to generalize to 3+ player or 3+ hand versions.

2 Data Sourcing and Processing

Our data comes entirely from simulation. Each training episode corresponds to a full Chopsticks game between two agents.

2.1 State Representation

We represent each state as $(p_{11}, p_{12}, p_{21}, p_{22})$, where p_{11}, p_{12} are the current player's hands and p_{21}, p_{22} are the opponent's hands. We sort each player's two hands so that the smaller value comes first. This reduces symmetry and cuts the state space to 225 states.

We use integer compression to map each state to a unique `STATE_ID`.

Throughout this report, "left hand" means the smaller-value hand and "right hand" means the larger one.

2.2 Actions

We define 8 canonical actions:

- **Attacks**: ATTACK_LL, ATTACK_LR, ATTACK_RL, ATTACK_RR
- **Swaps**: SWAP_L2R, SWAP_L1R, SWAP_R1L, SWAP_R2L

Attacks add hand values and normalize (kill dead hands, reorder hands). Swaps transfer 1–2 fingers between hands and normalize as well.

Because our representation always views the state from the perspective of the *current player*, each action also flips the state to make the opponent the new current player.

2.3 Transition Function

All transitions are deterministic: $T(s, a, s') = 1$ if legal, and 0 otherwise.

2.4 Reward

A reward of $+1$ is given only if the action immediately kills both opponent hands. Otherwise reward is 0. Despite seeming sparse, the modified Q-update (next section) makes this sufficient.

3 Modeling and Inference

We formulate Chopsticks as a two-player zero-sum MDP with deterministic transitions.

3.1 Q-Learning Algorithm and Parameters

Because after every action we flip perspective to the opponent, maximizing future reward would incorrectly give the opponent the best position. Thus our update instead *minimizes* the opponent's best future reward:

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

For terminal states, the update simplifies to:

$$Q(s, a) \leftarrow Q(s, a) + \alpha \left[r - Q(s, a) \right].$$

This behaves like a minimax-style Q-update but without explicit game-tree search.

After training, the agent consistently finds the known optimal strategy: if it plays second, it always wins (the theoretically perfect outcome).

4 Experiments and Analysis

We first used an initial learning rate of $\alpha = 0.01$ to train the model. Then, we experimented by varying the parameter $\alpha = 0.5, 0.1, 0.001, 0.00001$ to determine the best one. To do this, we kept track of the Q-learning convergence over trials, which is a measure of how much the Q-values change in one iteration of the training loop (using L2 diff). We also kept track of the average episode length over time, but this ended up being very similar across different α. Each model was trained with 10^5 iterations.

4.1 Graphs

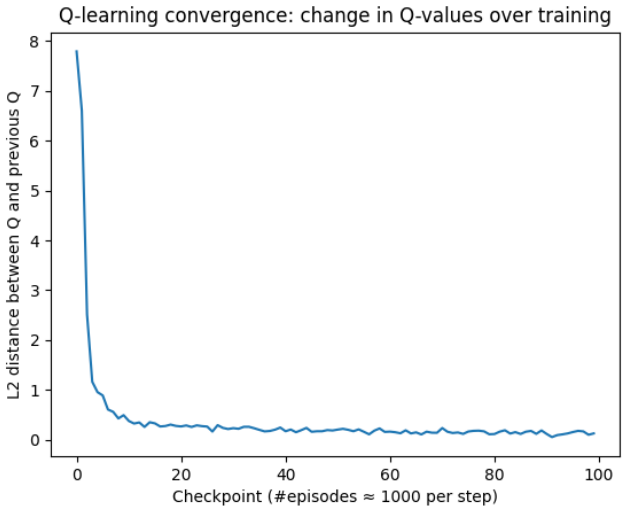

Figure 1: Q-learning convergence for $\alpha = 10^{-1}$

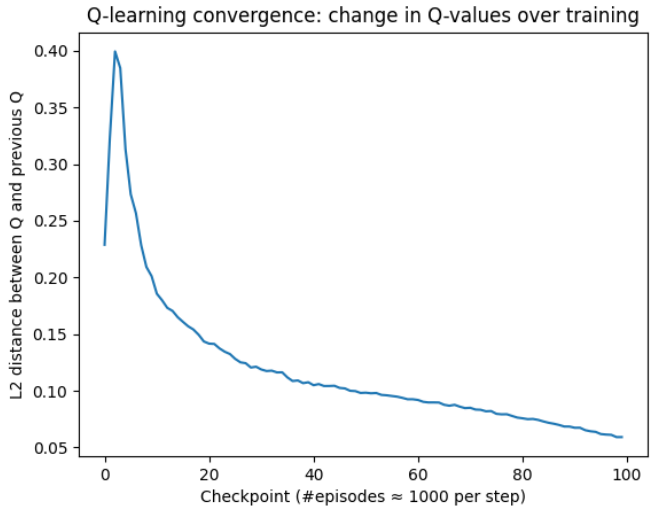

Figure 2: Q-learning convergence for $\alpha = 10^{-3}$

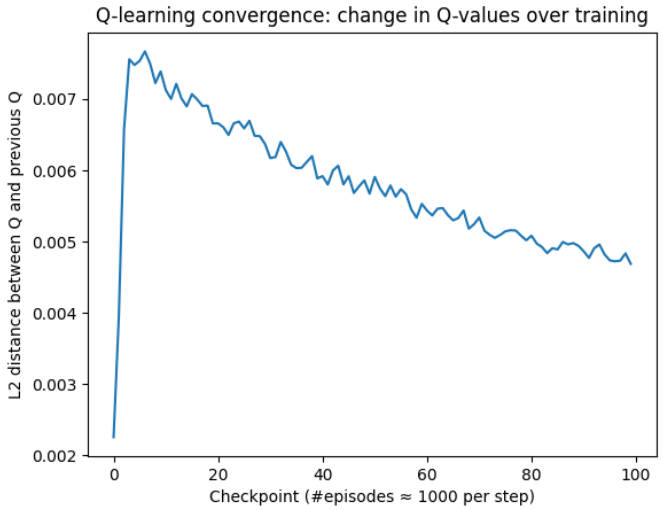

Figure 3: Q-learning convergence for $\alpha = 10^{-5}$

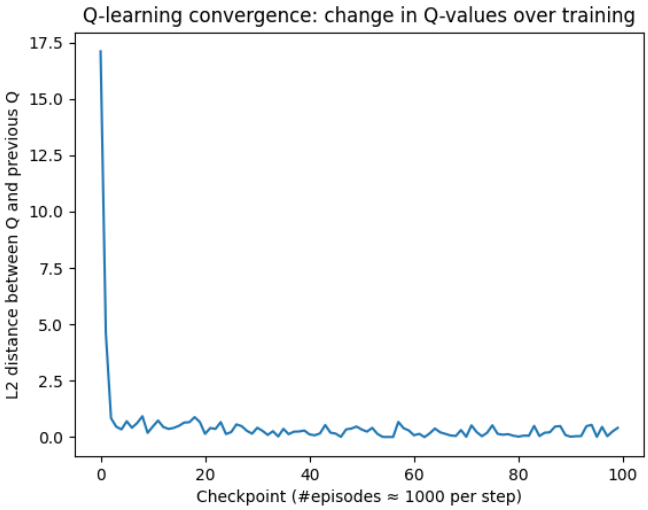

Figure 4: Q-learning convergence for $\alpha = 5 \times 10^{-1}$

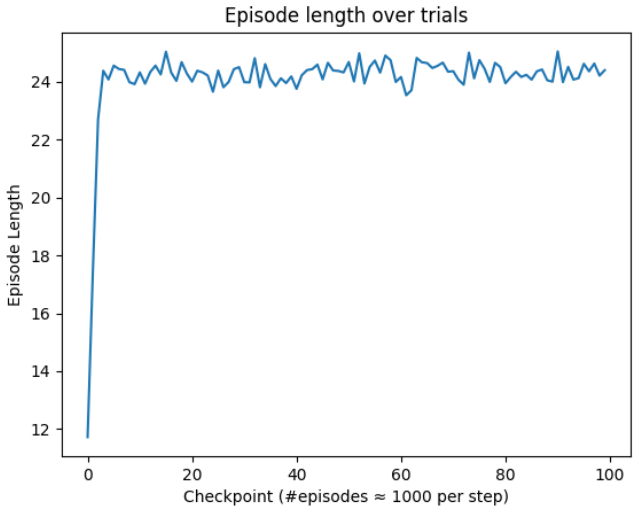

Figure 5: Episode length over training for $\alpha = 1 \times 10^{-2}$

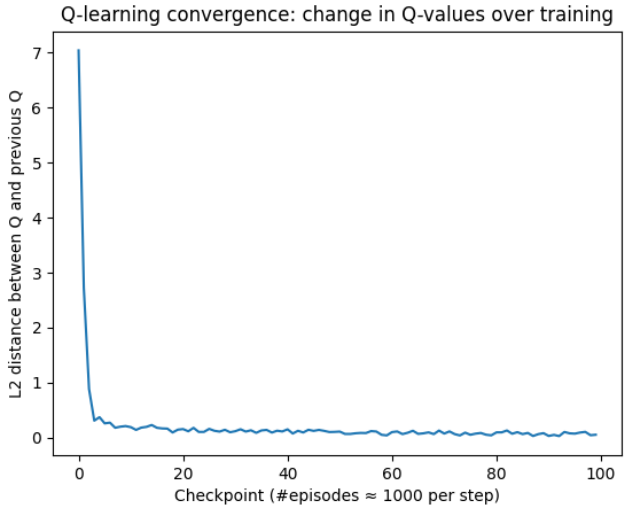

Figure 6: Q-learning convergence for $\alpha = 1 \times 10^{-2}$ (default)

4.2 Discussion

Across all tested learning rates α, the episode length remains roughly constant after the first few checkpoints. At the beginning of training, the agent often ends quickly before learning a reasonable strategy, yielding short episodes. After a few thousand episodes, behavior stabilizes.

The Q-value convergence graphs show the L_2 distance between successive Q-tables.

$\alpha = 10^{-5}$ **and** $\alpha = 10^{-3}$ Updates are extremely small, so the curves stay low. However, they do *not converge to zero* even after 100,000 episodes. Learning is too slow for convergence within the training horizon.

$\alpha = 10^{-2}$ This learning rate performs the best. Q-differences decrease rapidly and smoothly, with minimal noise at the bottom. This indicates stable and consistent convergence and that the agent has settled on a strong policy.

$\alpha = 10^{-1}$ Similar to 10^{-2} but with more variability near convergence. Still converges reasonably well, but updates are noisier because of the larger step size.

$\alpha = 5 \times 10^{-1}$ This learning rate is too large. Even though Q-differences initially fall quickly, the curve remains highly unstable and spiky. The agent frequently overcorrects and revises Q-values, preventing stable convergence.

Qualitatively speaking, the model (at $\alpha = 0.01$) performs quite well. It was able to find the optimal strategy for the game, which resulted in it always being able to win as the second player. The training of the model was computationally inexpensive, as we were able to run the entire training process in around 30 seconds.

5 Reflections and Contributions

Kai: I contributed to the project by helping to formalize the problem mathematically, writing milestone 1 2, and collecting graphs and metrics for the discussion section for milestone 3. This project taught me a lot about Q-Learning and the process of applying a learning algorithm to a problem. One thing I've learned from this project is how important it is to write everything out mathematically. Even though the problem may seem simple, by cleanly writing out things like states, actions, and rewards, the implementation of the code becomes much easier. Lastly, the project also helped me understand the difficulties in adding complexity to an existing problem. Adding 3 players to the chopstick game would make some of the optimizations we did invalid, and would require a completely different approach (multiple Q-Tables). I did not use GenAI for any of my contributions.

Arul: After working on this project, I've come up with some suggestions and advice to give others who want to recreate the project. Think carefully about how to choose a method to represent states and actions. It may be that in simple games such as chopsticks, there could be several states that seem different but actually have the same optimal action. Make sure you really understand the mechanisms behind what you are doing (like understanding the RL agent behavior, the motivation behind the bellman update, why even a simplistic reward function can actually do the job). In the long run it will actually save you a lot of time (like our modified update + terminal reward function did). We were able to make optimizations and cleaner code after we understood these topics better. Lastly, don't be too prideful to use AI. It probably has a much better understanding of a very common topic like RL, and you should use that to make sure you really get the underlying ideas. At the same time don't just pass code off to it, as even writing a basic RL bot for a simple game like chopsticks was clearly a little too hard for it.

Nikitha: I contributed to my team by helping brainstorm a concrete project idea for us, and I have also created, worked on, and submitted milestones 2 and 3. A suggestion I would give for students who are working on this project would be to constantly review the RL concepts you have learnt in class and keep them in mind as you are progressing through your project. This would help you make wiser decisions when it comes to finalizing state and actions, and understanding the feasibility/"do-ability" of certain features and next steps you would like to have in your game. This project has contributed a lot to my understanding in RL. It has put me in an environment when I have to directly interact with RL implementation for a game, which have help me understand that maybe RL is as scary as it looks on paper or the lecture slides. It has increased my interest is getting involved in similar if not more complicated projects in the future. I have used GenAI for certain parts of the milestones where I was stuck on how to format so much information in a readable manner and how to incorporate images within text. It has helped with my showing me some formatting shortcuts in Latex.

Karim: Out of all the contributions I made to this project, I would like to highlight that I helped plan and organize the work by meeting with my group members to discuss project ideas and researching potential topics to ensure they were both interesting and appropriate for the class. These efforts directly supported the progress we made in the Milestone deliverables.I also contributed to Milestone 2 by writing portions of the code myself and merging them with what my teammates had implemented. I tested my logic locally to compare results with my team and verify that the code behaved as expected. Running these tests helped me confirm that our approaches were aligned and that the overall logic of the project was functioning correctly. Overall, I had a great experience working with my team on this RL project. This was my first time diving deeply into reinforcement learning, and I found it exciting how RL resembles building a life-simulation system, except we must define the rules and logic ourselves for the computer to understand the environment. I've always struggled with theory and algorithmic reasoning, but applying the course concepts directly in this

project helped me understand RL much more clearly and learn how to translate theoretical ideas into working code.