Planning and Learning in *Polytopia*

Felipe Barbosa
Department of Computer Science
Stanford University
Stanford, CA 94305
`fbarbosa@stanford.edu`

Ion Martinis
Department of Computer Science
Stanford University
Stanford, CA 94305
`ion.martinis@stanford.edu`

Gabriel Noya
Department of Electrical Engineering
Stanford University
Stanford, CA 94305
`gkn@stanford.edu`

Abstract

We study decision-making of autonomous agents in *The Battle of Polytopia*, a turn-based strategy game that is uniquely complex for its uncertainty, large state-action space, and need for long-term planning. Modeling the game as a fully-observable MDP in the *Tribes* framework, we introduce **PolySolver**, a Monte Carlo Tree Search (MCTS) method that is enhanced with action filtering, heuristic rollouts, and a domain-aware value function. PolySolver achieves state-of-the-art performance, outperforming publicly available *Tribes* agents. To extend planning into learning, we further train a Deep Q-Learning (DQL) agent on PolySolver experiences. The resulting DQL agent is competitive against a strong MCTS opponent, showing it can recover a substantial portion of PolySolver's planning strength while following faster and more efficient strategies.

1 Introduction

Autonomous agents in turn-based strategy games must make decisions that influence long-term future game states. *The Battle of Polytopia* is one such game, where players expand territory, develop economies, and command units across a procedurally generated map. The game requires agents to model uncertainty, such as variable opponent behavior and battle outcomes, while also balancing immediate gains with long-term strategic development.

We build on *Tribes*, an open-source implementation of Polytopia equipped with a Forward Model to run lookahead simulations [Perez-Liebana et al., 2020]. Under simplifying game settings (e.g., full observability), we model the *Tribes* agent problem as a Markov Decision Process (MDP).

Tribes features multiple baseline AI agents, with Monte Carlo Tree Search (MCTS) being among the strongest. However, its default implementation leaves ample room for improvement. In this project, we introduce **PolySolver**, a Python-based MCTS agent designed to exceed the performance of baseline agents provided by *Tribes*. Beyond improving tree search, we further use PolySolver as a starting point for training a Deep Q-Learning (DQL) agent, enabling compact learning from PolySolver's cross-game experience.

Preprint.

2 Background and Appropriateness of Approach

2.1 *The Battle of Polytopia* as an MDP

The Battle of Polytopia is a turn-based strategy game in which players expand territory, fight enemies, develop economies, and command units across a 2D grid. Games end either by full elimination or after 30 turns, with the winner determined by score.

We use the open-source *Tribes* implementation of *The Battle of Polytopia*, which exposes a Forward Model (FM) for simulating actions and resulting states [Perez-Liebana et al. [2020]]. The problem of playing *Tribes* fits an MDP formulation, because gameplay involves sequential decisions under uncertainty and satisfies the Markov property through memoryless transitions. We define the MDP as such:

- **States:** Fully observable game configurations, including map layout, unit positions and health, city levels and production, researched technologies, remaining stars, opponent units and cities, scores, and turns remaining.
- **Actions:** One action in each decision point, chosen among economic (city upgrades, resource improvements), expansion (capturing villages, founding cities), military (spawning, moving, attacking), and technology (research). Actions are constrained by stars and prerequisite dependencies.
- **Transitions:** The *Tribes* Forward Model applies the selected action and returns the next state. The transition is stochastic from the agent's perspective due to variable opponent behavior, combat interactions, random ruin events, and procedurally generated map features.

Beyond modeling uncertainty, an agent in this game needs to have the ability for long-term planning. Early aggression can hinder economic growth, weak city development limits late-game production, and inefficient expansion strains military capacity. Short-sighted heuristics fail to capture these delayed dependencies. Modeling the game as an MDP provides the agent a framework to plan and learn over many turns.

2.2 Monte Carlo Tree Search (MCTS)

Monte Carlo Tree Search (MCTS) [Coulom, 2006] is a simulation-based planning method that estimates long-term action value by selectively exploring promising parts of the game tree. Each iteration consists of four steps: **Selection** (choose actions using an exploration/exploitation rule such as UCB1), **Expansion** (add a new child corresponding to an unexplored action), **Simulation** (roll out a trajectory to approximate future reward), and **Backpropagation** (update value estimates along the path). The UCB1 score for action i is

$$\frac{w_i}{s_i} + c\sqrt{\frac{\ln N}{s_i}}. \tag{1}$$

balancing exploitation (average payoff) with exploration (favoring less-visited actions).

MCTS is well-suited for the *Tribes* problem, which has huge branching factors on the order of 10^{15} for the full game [Perez-Liebana et al., 2020] Unlike brute-force methods (e.g., forward search), MCTS does not enumerate the full state space and can concentrate computation on high-value lines of play. Further, the *Tribes* Forward Model makes it inexpensive to simulate thousands of deep rollouts per decision, allowing MCTS to reason effectively about delayed rewards. Despite these strengths, MCTS performance depends heavily on rollout quality, evaluation heuristics, and action filtering, motivating the improvements introduced in our agent.

2.3 Deep Q-Learning

Deep Q-Learning (DQL) [Mnih et al., 2013] is an off-policy reinforcement learning method that approximates the optimal action-value function $Q^*(s, a)$ with a deep neural network. Instead of relying on forward simulation at decision time, as MCTS does, DQL learns long-term value estimates offline from experience. At inference time, the DQL agent selects the action that maximizes the parametrized $Q_\theta(s, a)$ for the current state.

Although tabular Q-learning would provide explicit value estimates, it is infeasible to use *Tribes* due to its enormous state space, which would lead to an intractably large Q-table and visiting many states sparsely. This motivates the need for DQL, which uses a function approximator that can generalize across state-action pairs.

In our approach, DQL learns from transitions generated by the PolySolver MCTS agent. Unlike MCTS, which has no memory and must rediscover good strategies in each game, DQL accumulates knowledge across games. As a result, the learned value function can support more informed decisions, even in previously unseen game situations. This is especially useful in *Tribes* where the rewards are delayed and many strategic patterns (e.g., city development, economy management, army positioning) recur across games. In addition, once trained, DQL produces action-value estimates with a single forward pass, which makes decisions effectively constant-time despite the game's large branching factor.

3 Methods

3.1 PolySolver MCTS

3.1.1 Action Strategy

During the selection phase of MCTS, we design our agent to expand only to the most promising branches by imposing action prioritization and dynamic scaling.

We group actions into three priority tiers: High (`SPAWN`, `ATTACK`, `CAPTURE`), Medium (`MOVE`, `BUILD`, `RESEARCH`), and Low (`END_TURN`, `DECLARE_WAR`). From our intuition of the game, the high-priority actions tend to lead to more meaningful tactical change than the low-priority actions. To reduce unproductive branching, we prune actions such as `SEND_STARS` and `END_TURN`, to give the opportunity to explore other actions first before ending the turn.

Because the action set can grow to 30-50 actions as the game progresses, we limit branching using the following dynamic threshold for N applied in the UCB1 formula (Formula 1):

$$N = 20 + 3\sqrt{s_i + 1} \tag{2}$$

This makes it so that states with few visits explore only a small subset of actions, whereas frequently visited states receive more computational budget. It concentrates search effort around promising regions and keeps weak branches shallow.

3.1.2 Rollout Strategy

During simulation, our PolySolver agent performs rollouts of up to 100 actions, corresponding to approximately 10-15 turns of *Tribes*. This depth was selected to balance computational feasibility with the ability to foresee impactful strategic developments such as city captures, tech progress, and multi-turn combat.

Unlike *Tribes*'s MCTS implementation, which uses random rollouts, our PolySolver agent uses a heurisitcs-guided rollout policy. Each candidate action is scored for its tactical value: +200 for `CAPTURE`, +80 for `SPAWN`, +50 for `ATTACK`, +100 for killing an enemy unit, -50 for `END_TURN`, +0-10 as random noise, +50000 for a win, -50000 for a loss. At each rollout step, the action with the highest heuristic score is selected. This ensures that the simulated trajectories are biased towards success by emphasizing expansion, city pressure, and decisive combat rather than drifting into passive play. Rollouts also terminate early if the game ends, ensuring efficient simulation.

3.1.3 Reward Function

Although the rollout heuristic selects locally good actions, the backpropagation phase relies on a more comprehensive evaluation function at the final rollout state. This function is designed to reflect long-term strategic advantage relevant to *The Battle of Polytopia*.

We define the value of a final rollout state as:

$$\begin{aligned} R = {} & 50000(\mathbf{1}_{\text{win}} - \mathbf{1}_{\text{loss}}) + \text{myScore} + 20000(2\,\text{myCities} - \text{oppCities}) \\ & + 5000(2\,\text{myUnits} - \text{oppUnits}) + 2000\,\text{threatBonus} \\ & + 100(\text{myProduction} + 1000 - \text{oppTotalHealth}) + 500\,\text{myTechs}. \end{aligned} \tag{3}$$

Formula 3 rewards city control, military advantage, technological development, and direct pressure on enemy cities, which are features strongly correlated with eventual victory in *The Battle of Polytopia*.

The function combines both absolute and differential metrics so that the agent values not only improving its own position but also weakening the opponent. The value function also helps prevent degenerate strategies such as stalling for score.

3.1.4 Backpropagation

In backpropagation, the terminal rollout value R is propagated from the terminal state back to the root, updating the visit count and cumulative value of each intermediate state v:

$$N(v) \leftarrow N(v) + 1, Q(v) \leftarrow Q(v) + R. \tag{4}$$

We have configured the evaluation function to produce large-magnitude values. Over many simulations, early states with higher strategic importance naturally emerge with higher Q values, guiding the agent toward policies involving those states.

3.2 Deep Q-Learning (DQL)

3.2.1 Generation of Training Data

To train the Deep Q-Learning agent, we generate a dataset of transitions from 80 full games played between our PolySolver MCTS agent (set at 1500ms decision time budget) and a range of baseline opponents. Using PolySolver as the teacher instead of a random agent allows DQL to learn from transitions that are already strategically meaningful.

For every decision made by PolySolver, we record full transition tuples (s, a, r, s'). Including the next state s' is essential for applying the Bellman optimality equation (Formula 5), which defines the target for temporal-difference (TD) learning. To expand coverage, we also include additional transitions for actions that were legal but not selected by PolySolver. This increases the diversity of the training set and helps the learned Q-function generalize across the full action space.

3.2.2 DQL Training

Given the collected transitions (s, a, r, s'), the goal of DQL is to approximate the optimal action-value function $Q^*(s, a)$. We train a 2-layer MLP that takes as input the state/action-feature encodings, and outputs a scalar estimate of $Q^*(s, a)$.

Each transition is used as a training example to estimate the target Q-value y using the Bellman optimality equation

$$y = r + \gamma \max_{a'} Q_{\theta^-}(s', a'), \tag{5}$$

where $\gamma = 0.99$ is the selected discount factor and Q_{θ^-} is a periodically updated target network.

The weights θ are optimized to minimize the mean-squared error between the predicted value $Q_\theta(s, a)$ and target y. We train the MLP for 50 epochs over the entire dataset, with learning rate 10^{-5} and mini-batches of size 128. To help stabilize training, the data points are sampled uniformly from a replay buffer, in order to de-correlate consecutive state transitions and reduce overfitting.

4 Results

4.1 PolySolver MCTS

We evaluated the PolySolver agent in one-on-one games against baseline AI agents implemented for *Tribes* by Perez-Liebana et al. [2020], which remain the only publicly available agents for this environment. We tested performance against three opponent agents:

(1) Random Agent uniformly samples from legal actions.
(2) One-Step Lookahead Agent (OSLA) applies each action in the Forward Model, values the next state, and chooses the highest-value action.
(3) MCTS Agent uses the standard selection-expansion-simulation-backpropagation with the *Tribes* framework's pruning rules, random rollouts, and default evaluation. For fairness to the MCTS agents, we set identical rollout depth (100 actions), exploration constant ($\sqrt{2}$), and time budget (150 ms).

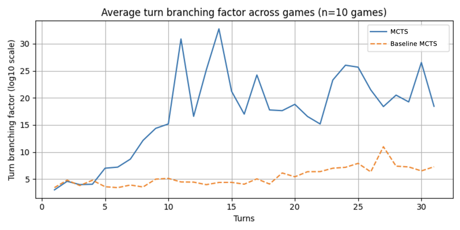
All games were run in Score mode. Games ended by elimination or after 30 rounds, with higher score determining the winner. Each matchup consisted of 20 games, with starting tribes alternating to account for assymetry in initial positions. Results are shown in Table 1.

Opponent	Win Rate	Avg Score Diff	Avg Cities Diff	Avg Units Diff	Avg Turns
Random	100%	3321.5	2.00	8.90	27.2
OSLA	100%	2837.4	0.45	4.40	30.0
MCTS	100%	2569.3	1.95	9.60	27.4

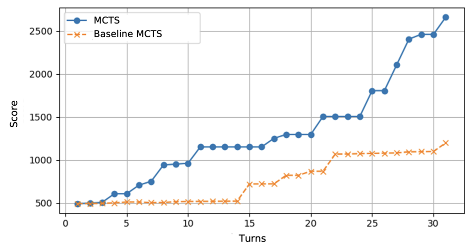
Table 1: Performance of PolySolver MCTS against baseline agents (n=20).

PolySolver MCTS had superior performance against baseline *Tribes* agents. Against all three baselines, PolySolver achieved a 100% win rate, with large margins in score, board control (cities), and military strength (units). Most games lasted the full 30 turns, indicating PolySolver's sustained strategic dominance rather than early eliminations. Among the baselines, PolySolver had the smallest score difference with baseline MCTS, followed by OSLA then Random, which is consistent with our intuition that forward planning is ideal for performance in *Polytopia*.

It is notable that PolySolver consistently outperforms the baseline *Tribes* MCTS, despite both algorithms following the MCTS framework and being constrained to identical search parameters. To further understand this performance difference, we collected metrics for the agents' decision-making. Throughout the games, PolySolver takes on average substantially more actions per turn (33.0 actions vs. 14.5 actions for baseline MCTS). This indicates that PolySolver ends turns later and therefore can explore/exploit a larger portion of the action space early on. Next, to study the size of the action space available, we computed the branching factor per turn (product of $|A|$ across decisions in the turn, where $A =$ set of available actions). Results are shown in Figure 1a averaged across 10 games. It is expected that for both agents, the branching factor increases with more turns, because the board becomes incrementally more complex and unlocks new actions for the agent. While both agents initially have a similar branching factor (turns 1-4), branching then increases much more sharply for PolySolver than for baseline MCTS. Actions made early in the game appear to allow PolySolver to access a more complex board mid-game, so PolySolver has a higher optionality of actions from which to optimize.



(a) Branching factor per turn averaged across 10 games.



(b) Sample agent score per turn.

Figure 1: Comparison of branching factors and score progression between Polysolver and baseline MCTS.

We attribute the performance difference to two design choices. First, PolySolver's action strategy prioritizes aggressive actions (e.g., attacks, captures, spawns) and deprioritizes premature end-turn actions. This could be why PolySolver takes, on average, more actions per turn. PolySolver's smarter action selection could allow it to focus its budget on meaningful branches. Second, PolySolver's state evaluation function could be a more informative proxy for long-term success. PolySolver explicitly rewards reducing the opponent's health, while baseline MCTS does not. This could explain why in the first few turns - before the board becomes complex and action space differences dominate - PolySolver already exceeds the score of baseline MCTS (Figure 1b).

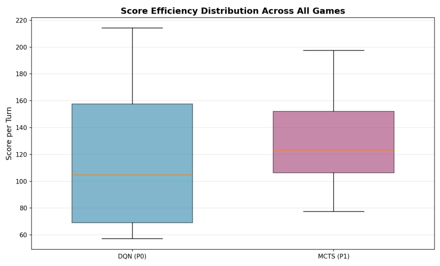
4.2 Deep Q-Learning

We evaluated the DQL agent by playing it against PolySolver MCTS, set to a fixed 600 ms decision-time budget. The results are shown in Figure 2 a-b.

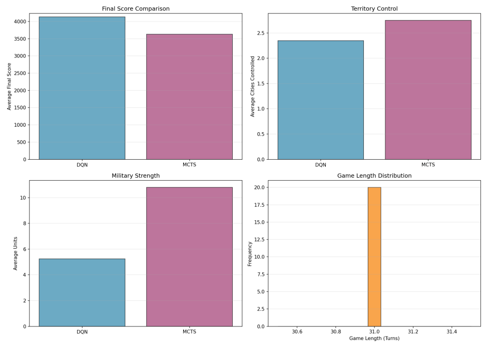
Across 20 head-to-head games, the DQL agent achieved a **60%** win rate, indicating that it recovers a substantial portion of PolySolver's planning strength despite performing no search at inference time.

Although MCTS achieved higher average final score and maintained stronger board presence, controlling more cities and producing roughly twice as many units (Figure 2b), the DQL agent remained competitive in the strategic trajectories that mattered for winning. Interestingly, the DQL agent also earned a lower score per turn on average (117 vs. 130 for MCTS; Figure 2a). MCTS tends to generate more economic output even in games it ultimately loses, while DQL wins tend to be more decisive with lower economic output. This suggests DQL can be successful in *Tribes* while following more efficient strategies.

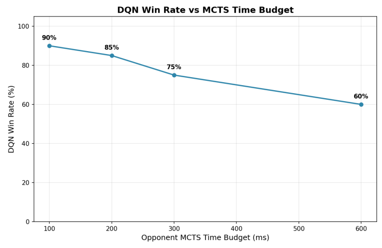
To assess robustness, we further varied the decision time of PolySolver from 100 ms to 600 ms. As shown in Figure 2c, the DQL win rate decreases smoothly from 90% against 100 ms MCTS to 60% against the 600 ms version. This trend is expected. Deeper MCTS search reveals strategic opportunities that a fixed-value function cannot always match. That said, DQL remains competitive even against substantially more computationally powerful planners. Last, the training and validation losses (Figure 2d) converge steadily over the first ~ 15 epochs, suggesting that the deep neural network successfully internalized value structure from MCTS-based trajectories.



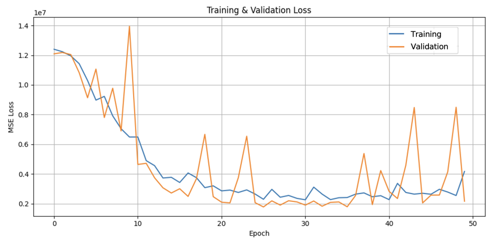
(a) Score per turn distribution.



(b) Avg. score, cities, units.



(c) Win rate vs. MCTS time budget.



(d) Training and validation loss.

Figure 2: Deep Q-Learning evaluation results against PolySolver MCTS. Panels (a)–(b) show head-to-head performance at a 600 ms MCTS budget; (c) shows how the DQL win rate varies with the opponent's search time; and (d) shows DQL training convergence.

5 Discussion & Conclusion

Our work builds on the *Tribes* framework, which established a forward-model planning environment for *The Battle of Polytopia* and introduced a baseline MCTS agent Perez-Liebana et al. [2020]. While the baseline MCTS suggests that simulation-based planning can outperform heuristic baselines, it leaves significant room for improvement.

We introduce PolySolver, an enhanced MCTS that achieves consistent and substantial gains over baseline agents. PolySolver wins 100% of games against all provided baselines, including the original *Tribes* MCTS, and dominates across score, city control, and military production. These results suggest that introducing lightweight, domain-aware structure into MCTS can dramatically improve search quality without increasing computational budget. By filtering the action space to prioritize impactful actions and suppress premature end-turn actions, and guiding rollouts with heuristics, PolySolver focuses its simulations on strategically meaningful branches of the tree. These early decisions lead to richer mid-game positions, as reflected by PolySolver's higher mid-game branching factor. Importantly, PolySolver is successful against baseline agents despite deciding in strict 150 ms time budgets. Allocating more time could give further upside in performance. A key limitation of both baseline MCTS and PolySolver is that they assume random/passive opponent behavior in rollouts. This simplification could be causing both MCTS agents to undervalue counterattacks and occasionally overcommit to aggressive lines. More sophisticated opponent modeling remains a direction for improvement.

Beyond planning, we demonstrate that learning from PolySolver's game experience is a promising approach for producing fast, search-free agents. We introduce a DQL agent, trained on trajectories from a 1.5 s version of PolySolver, that won 60% of games against the optimized 600 ms PolySolver. This shows that a neural value function can recover a substantial portion of a planner's strategic competence, even in environments with large branching factors and long-term dependencies. However, DQL results also highlight important challenges. Performance degrades smoothly as the opponent's time budget increases, reflecting the difficulty of imitating a planner in positions requiring far lookahead. The learned policy is also sensitive to mismatch between the MCTS trajectories used for training and the opponent it encounters during evaluation.

Overall, we demonstrate a complementary relationship between online planning and learning in *The Battle of Polytopia*. With PolySolver, we show that domain-informed enhancements of MCTS can improve planning without increasing computational budget. With DQL, we show that parts of this behavior can be captured in a compact neural policy. Future work should explore tighter integration of these approaches, such as policy distillation, learned heuristics for guiding MCTS, or iterative bootstrapping through self-play. It should also extend agents to applied settings, such as play against human opponents and *Polytopia* environments with partial observability.

6 Group Contribution

- Felipe Barbosa: MCTS model, DQL finetuning, *Tribes* integration, work on introduction, background, methods, and conclusion
- Ion Martinis: MCTS benchmarking, DQL model, work on abstract, introduction, background, methods, and results
- Gabriel Noya: MCTS finetuning, DQL finetuning, DQL benchmarking, work on results and conclusion

The development of Deep Q-Learning was our extension for four units.

References

Rémi Coulom. Efficient selectivity and backup operators in monte-carlo tree search. *CG 2006*, 2006.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin A. Riedmiller. Playing atari with deep reinforcement learning. *CoRR*, abs/1312.5602, 2013. URL `http://arxiv.org/abs/1312.5602`.

Diego Perez-Liebana, Yu-Jhen Hsu, Stavros Emmanouilidis, Bobby Dewan Akram Khaleque, and Raluca D. Gaina. Tribes: A new turn-based strategy game for ai research. In *Proceedings of the Sixteenth AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment (AIIDE)*, 2020.