MULTI-AGENT TIC-TAC-TOE TOURNAMENT

CSCI-6660: Intro to AI

Aryan Tandon, Bipin Kumar Thollikonda

WHAT WE BUILT

- 5 different AI agents that compete in a round-robin tournament.
- Each agent plays the other agent twice (once as X, once as O)
- 200 games per matchup (4,000 total games)
- **Goal:**
  - *Learn how different AI paradigms (search-based, rule-based, learning-based, simulation-based) work*
  - *Compare performances and see which strategies work best and why?*

MEET THE AGENTS

- Random: Baseline – Pure chance, no strategy
- Heuristic: Rule-based – Human-like decision patterns
- Minimax (Alpha-Beta): Search-based - Perfect play through exhaustive search
- Q-Learning: Learning-base - Discovered strategy through 100,000 self-play games
- MCTS: Simulation-based: Statistical sampling of possible futures

Random Agent

- **Strategy:** Pick any legal moves randomly
- **Intelligence Level:** None
- **Odds:** Underdog in all matchups
- **Code Concept:**

```python
def get_move(self, game):
    """...
    legal_moves = game.get_legal_moves()
    return random.choice(legal_moves)
```

Heuristic Agent

- **Strategy:** Follow priority rules:
  - *Win if possible*
  - *Block opponent's winning move*
  - *Take center if available*
  - *Take corner*
  - *Take any remaining move*
- This Agent is fast, makes human-like decisions, but doesn't plan ahead
- **Odds:** Favorites against most opponents

```python
def get_move(self, game):
    """...

    board = game.board

    winning_move = self.find_winning_move(board, self.player)
    if winning_move is not None:
        return winning_move

    opponent = -self.player
    blocking_move = self.find_winning_move(board, opponent)
    if blocking_move is not None:
        return blocking_move

    if board[4] == 0:
        return 4

    corners = [0, 2, 6, 8]
    for corner in corners:
        if board[corner] == 0:
            return corner

    edges = [1, 3, 5, 7]
    for edge in edges:
        if board[edge] == 0:
            return edge

    return game.get_legal_moves()[0]
```

Minimax Agent

- **Strategy:** Explore every possible game outcome
  - *Recursively builds game tree by trying every legal move*
  - *Assumes opponent plays optimally*
  - *Chooses move that guarantees best outcomes*
- **Alpha-Beta Pruning:** Eliminates ~96% of unnecessary branches
- **Result:** Mathematically unbeatable in tic-tac-toe
- **Odds:** Heavy Favorite in every matchup

```python
def get_move(self, game):
    """...

    best_score = float('-inf')
    best_move = None
    alpha = float('-inf')
    beta = float('inf')


    for move in game.get_legal_moves():
        game_copy = game.copy()
        game_copy.make_move(move)

        score = self.minimax(game_copy, False, alpha, beta)

        if score > best_score:
            best_score = score
            best_move = move

        alpha = max(alpha, best_score)

    return best_move
```

Q-Learning Agent

- **Strategy:** Learn through experience
  - *Plays 75,000 games against itself*
  - *Starts with zero-knowledge of Tic-Tac-Toe*
  - *Tries moves, observes outcomes*
  - *Builds Q-Table: state -> action -> expected value*
  - *Balances exploration vs exploitation*
- **Odds:** Wildcard – (could dominate or disappoint based on its training)

```python
def get_move(self, game, training=False):
    """...

    state = self.transform_state(game.board)
    legal_moves = game.get_legal_moves()

    # Exploration vs Exploitation
    if training and random.random() < self.epsilon:
        move = random.choice(legal_moves)
    else:
        move = self.get_best_move(state, legal_moves)

    if training:
        self.history.append((state, move))

    return move
```

Monte-Carlo Tree Search (MCTS) Agent

- **Strategy:** Statistical sampling of possible futures
  - *For each possible move:*
    - Simulates 1000+ random games
    - Track win/loss/draw rates
    - Chooses move with highest success rate
- Balances exploration of new moves vs exploitation of known good moves
- Computationally the most expensive during gameplay
- **Odds:** Solid contender but not unbeatable

Tournament Design

- **Format:** Round-robin (every agent plays every other agent)
- **Games per matchup:** 200 games
- First player (X) has advantage; therefore, each matchup alternates the starting position (X and O)
- **Total games:** 4,000 games across all matchups
- **Metrics Tracked:** Wins, losses, draws, performance by position

Live Tournament Demo

Tournament Results

FINAL STANDINGS

Rank	Agent	Points	W	D	L	Games	Win%
1	Minimax	2173/3200	573	1027	0	1600	35.8%
2	Heuristic	2157/3200	558	1041	1	1600	34.9%
3	MCTS	2012/3200	416	1180	4	1600	26.0%
4	Q-Learning	1536/3200	370	796	434	1600	23.1%
5	Random	122/3200	6	110	1484	1600	0.4%

Key Finding #1

- Perfect plays usually ends in draw
  - *Minimax vs Heuristic (100% draws)*
  - *MCTS vs Heuristic (99.5% draws)*
  - *Minimax vs MCTS (99.5% draws)*
- When both agents play near-optimally, winning becomes impossible
- **Takeaway:** Intelligence doesn’t result in more wins; it results in fewer losses against weaker opponents

Key Finding #2

- Heuristic never lost to Minimax (100% draw rate)
- Heuristic uses basic if-then rules with no calculation
- Minimax calculates every possible game outcome
- Both achieved identical performance
- Heuristic ran 50x faster (0.001s vs 0.05s per move)
- **Takeaway:** Sometimes the simplest solution is the smartest solution

Key Finding #3

- Q-Learning never won a single game against Minmax, Heuristic, or MCTS
- 75,000 training games were not enough
- Learned effective defensive play (50% draw rate against top agents)
- Couldn’t learn offensive plays
- **Takeaway:** Reinforcement learning needs more than volume, it needs better training structure and reward shaping

Key Finding #4

- MCTS achieved 99.8% unbeaten rate by running 1000+ simulations per move
- MCTS Crushed Q-Learning (29 wins for MCTS, 1 for Q-Learning)
- MCTS thinks fresh for every move it sees
- Q-Learning relies on pre-trained patterns
- **Takeaway:** Good approximation beats imperfect learning

Key Finding #5

- Three distinct Tiers emerged
- Tier 1 (Minimax, Heuristic) : Nearly perfect, ~35% win rate, 0-1 losses total
- Tier 2 (MCTS): Very strong, ~26% win rate, 4 losses total
- Tier 3(Q-Learning): Competent but flawed, ~23% win rate, 434 losses

Key Findings #6

- Random beat Heuristic 1 time out of 400 games.
- It happened because random by luck created a fork (two winning threats simultaneously)
- Heuristic could only block one threat resulting in Random Win
- Random couldn't win against Minimax because Minimax does exhaustive search, hence it doesn't allow a fork to ever be created.

Thank You!