Monte Carlo Tree Search for the Pokemon Trading Card Game

Tyler Uhlenkamp   TAUHLENK@IASTATE.EDU

Iowa State University

Tyler Chenhall   TCHENHAL@IASTATE.EDU

Iowa State University

Abstract

The Pokemon Trading Card Game is a popular turn-based card game usually played by two players. Its partially observable, stochastic, multi-agent nature combined with its complex rules, large state space, and demand for long-term planning make it an interesting target for modern artificial intelligence research efforts. An agent based on a Monte Carlo Tree Search (MCTS) algorithm with added knowledge via a heuristic function was developed and tested in games of PCG against human players and other agents. Although the MCTS agent performed well against random agents, its performance against human players was worse than expected. Possible explanations for this performance gap are explored and suggestions for future research and technical enhancements are proposed.

1. Introduction

The Pokemon Trading Card game has several characteristics that make it an interesting problem in artificially intelligent agent design. The general format and objective of the game, as well as its formal environment characterization and other important characteristics are summarized below.

1.1. Background: The Pokemon Trading Card Game

The Pokemon Trading Card Game is a collectible card game inspired by the Pokemon TV show originating in the early 1990s. Games are typically played by two people, each of whom uses a separate deck of 60 cards. Designing a deck involves significant planning and strategy which is separate from game-play, and so we do not discuss the de-

Com S 472/572 Final Project Report

tails in this document. Each player's deck consists of several types of cards: Energy cards, Trainer cards, and Pokemon cards. Pokemon cards represent the creatures from the popular TV show, and also represent the main players in the card game. A person plays Pokemon cards throughout the game, and attempts to strengthen them while doing "battle" in a turn-based format. Each Pokemon has several characteristics, including hit points, weaknesses, resistances, and attacks. Some Pokemon can also evolve to become stronger or more useful. Trainer cards serve as helpers in the game, often aiding to acquire cards needed for various strategies, or "healing" a Pokemon. Finally, Energy cards are needed to use certain actions such as using an attack or switching which Pokemon is currently active.

Game-play for the Pokemon card game is fairly complex. At the beginning of the game, each player draws seven cards from their shuffled decks. Re-drawing if necessary, they select basic (non-evolved) Pokemon to place in some of their six active Pokemon slots, one of which is considered "active". Then, each player deals out six face-down prize cards to set aside. These are collected one-by-one whenever a player defeats an opponent's Pokemon.

After set-up, the game proceeds in turns. During a single player's turn several actions occur. Each turn begins with the current player drawing a card. The person may then play one or more cards from their hand, including up to one Energy card, and one Supporter Trainer card. Pokemon evolution cards can also be used to evolve those currently on the field as long as that card has been in play for at least one turn. The player may also choose to switch their active Pokemon by paying its retreat cost (discarding a number of Energy cards from the active Pokemon). These actions may carried out in any order. The turn ends either when the player chooses to end their turn or upon choosing to attack with their active Pokemon. Note that Trainer cards are typically discarded after their effects apply, while Pokemon and Energy cards remain in play until the Pokemon takes more damage than it has hit points. Energies are typically not discarded for an attack unless otherwise specified by a

specific card.

How does the game end? There are three ways to win. First, if a player has no cards remaining in their deck when their turn starts, their opponent wins immediately. A player also loses if their last in play Pokemon is defeated, or when their opponent claims their sixth and final prize card.

It is important to note that card effects in the Pokemon game are both diverse and disparate in their behavior. Some Pokemon attacks will simply apply damage to the opponent's active Pokemon, while others do damage to many or all of them. Other attacks apply status effects with compounding results, heal, or even cause an opposing Pokemon to de-evolve. Trainer cards also have a variety of effects, many of which allow the player to search their normally unseen deck in order to select one or more cards of their choice. These effects contribute to the complexity of the game, both in strategy and representation.

1.2. Summary: Game Environment

Several characteristics of the game environment pose challenges when attempting to develop an intelligent computer agent to play the game well. At any given point in time, a player cannot typically see the cards in their opponent's hand, deck or prizes. Similarly, they cannot usually see their own prizes or deck. On the other hand, all previously played cards (discard piles) are visible, as are all cards currently in play. This indicates that the game state is only partially observable. The Pokemon game also has stochastic elements, including coin flips, shuffling of the deck, and card draws. As a two-player game with only one winner (draws are rare), it is also necessarily an example of a multi-agent environment. Finally, at any given point in a turn, the player may have numerous actions available, such as switching, attacking, playing cards from their hand, or simply ending their turn. This often results in a large branching factor, even before considering possible stochastic effects. Good human players are often able to concoct complex strategies combining the effects of multiple actions for maximum benefit.

Fortunately, some aspects of the card game lend themselves to standard analysis techniques. Each action in the game is executed sequentially, and the state is both static and discrete; the game state will remain the same even if an agent requires some time to select a move. The main limiter of agent analysis time is just the patience of any human opponents or observers.

In summary, playing Pokemon well requires long-term planning that accounts for a large and uncertain game tree.

2. Methodology

In an effort to develop an agent which could play the Pokemon Card Game competently against human players, the developers implemented a custom state representation and game logic from scratch, then created an agent based on a Monte Carlo Tree Search (MCTS) and a well-tuned heuristic function.

2.1. Game Logic and State Representation

All code representing game logic and state representation was written in Java specifically for this project. At any given point in time, the game state consists of a variety of information. This includes which cards are in both players' hands, discard piles, and prizes, as well as the order of cards in each deck. Each player can have up to six Pokemon in play, each with Energy attached. There are also temporary or persistent attributes to keep track of, such as damage done to each Pokemon so far, status effects on active Pokemon, whether an Energy or Supporter Trainer card has been played in a given turn and so on. Due to the large volume of information involved, the developers created a custom State data structure consisting of two identical objects representing the two players' cards, along with the previously mentioned attributes. The State can be cloned for use in agent decision making.

Cards are represented using a collection of 3 subclasses of the Card class: Energy, Trainer, and Pokemon. A single card is created by reading from a plain-text database which stores vital details as a list of attributes with some shorthand effect notation. For example, the Pokemon card "Hydreigon" has an attack called Consume. This attack requires 1 Dark, and 2 colorless Energy to use, has a base damage of 40 and heals the active Pokemon by as much damage as it does to the opponent. This attack is represented in the database as: `1D2C,Consume,doDamage:40,healDamage:EQ`. This format was chosen to avoid needing to write a new class for every single card, the convenience of brevity, and the ability to define a single "effect handler" which can parse and interpret the effects of complex, multi-faceted attacks.

At any given point in time, the code is also able to generate an exhaustive list of available commands. The same text-based command syntax is shared between human plays (input via command line) and computer agents

2.2. The Agent Program

The Monte Carlo Tree Search algorithm was chosen as the basis of the Pokemon Card Game agent because the algorithm has seen wide success in a variety of related games such as Poker, Settlers of Catan, and the "Magic: The Gath-

ering" card game. Furthermore, agents based on planning or rule-based approaches wouldn't be robust enough to create new strategies for brand new decks. The minimax algorithm also proves to be an undesirable candidate due to the partially observable and stochastic nature of the game environment as well as the large branching factor and need for long-term planning. The MCTS algorithm, on the other hand, has been applied to similar games and can be easily adapted to partially observable and stochastic environments. It also can exhibit long-term planning and works well in environments with a large branching factor, such as Go.

2.2.1. MONTE CARLO TREE SEARCH IMPLEMENTATION DETAILS

A full description of the MCTS algorithm and its properties is left to previous research while a summary of the implementation used in the PCG agent is provided.

The MCTS algorithm works by building a search tree node-by-node and estimating the effectiveness of each node by simulating many games which visit that node and keeping track of the expectation of reward.

The MCTS agent developed for the PCG agent maintains a persistent game tree structure which is saved between turns. At the beginning of a turn, the current state is located in the current game tree and set as the new root node. In this way, results of simulations in previous turns can still provide information about the usefulness of nodes still under consideration. Next, the MCTS algorithm, consisting of Selection, Expansion, Simulation, and Back-propagation stages (described below) are repeated in a loop for up to five seconds. Each iteration may add up to one new node to the search tree and simulates one play-out to the end of the game. Usually, the PCG agent was able to complete 15,000 such iterations per turn on a consumer-grade laptop computer. Finally, the child of the root node with the highest computed value is chosen as the next action.

Typical MCTS implementations assume uniform node types, however the PCG bot employed a unique tree structure in order to account for the stochastic nature of actions at each step in game play. Namely, it is assumed that each action results in one or more possible states and the state which is actually reached is the result of random chance. Therefore, the game tree consists of alternating levels of min/max nodes and chance nodes. A min/max node seeks to maximize or minimize the value of its children, while chance nodes randomly result in one of their children.

Each iteration of the MCTS algorithm consists of four stages as noted previously: Selection, Expansion, Simulation, and Back-propagation. The implementation of each stage is described below.

Selection: Starting from the root node, "optimal children are selected recursively until a node which is a leaf node or a node which still has un-expanded children is encountered. When selecting children, one must be careful to balance the exploitation of nodes which are known to be favorable to the current player and nodes which have not been explored as much. To accomplish this goal, the Upper Confidence Bounds formula was used. Namely, at each step the child node which maximizes the following equation was chosen:

$$v_i + C \times \sqrt{\frac{\ln N}{n_i}}$$

where v_i is the value of the current node,

2.2.2. HEURISTIC FUNCTION

Describe the heuristic function here

3. Results

Some stuff about how we tested

3.1. Test Results

Results vs random AI, observations vs humans

3.2. Explanations and Future Work

4. Conclusion