General-Purpose Planning Algorithms in the Card Game *Duelyst II*

Bryan McKenney
Department of Computer Science
University of New Hampshire
Durham, New Hampshire, USA
Email: bryan.mckenney@unh.edu

Wheeler Ruml
Department of Computer Science
University of New Hampshire
Durham, New Hampshire, USA
Email: ruml@cs.unh.edu

***Abstract*—*Duelyst II* is an online collectible card game (CCG) that features a 9x5 grid board, making it a cross between the popular CCG *Hearthstone* and chess. It is a partially-observable stochastic game (POSG) with a large branching factor and the ability to take several actions in a time-limited turn, making it a challenging domain for AI. The existing "starter AI" in the game is an expert-rule-based player that is limited to using certain decks and is weak against humans. We develop simple general-purpose planning algorithms that are able to consistently beat the starter AI using little domain knowledge and no learning. The most complex of these is a variant of Monte Carlo tree search (MCTS), for which we show that a novel action factoring method is helpful under certain conditions.**

I. Introduction

Duelyst II (henceforth referred to simply as "Duelyst") is a two-player online collectible card game (CCG) that features a 9x5 grid board where unit cards can move around and fight. Each player's goal is to keep their General alive and kill the enemy General using the resources in their deck.

Duelyst is an unexplored game in AI research. Creating an AI to intelligently play Duelyst is a challenge because there is a large branching factor (often around 100), several actions can be taken per turn within a 90-second limit, actions can be stochastic, and the opponent's deck and hand are unknown. Formally, it is a zero-sum partially-observable stochastic game (POSG). The existing AI in Duelyst, called the "starter AI," is an expert-rule-based player that is limited to using certain decks and is weak against humans. There are many ways in which a smarter AI that can play any deck could improve the player experience and introduce new content to the game. In this paper, we describe general-purpose planners, including a modified version of Monte Carlo tree search (MCTS), that are able to consistently beat the starter AI.

In the following sections, we discuss relevant work on games, describe Duelyst and the starter AI, introduce our variant of MCTS, explain our static evaluator, present experimental results, and, lastly, consider limitations and possible extensions.

979-8-3503-2277-4/23/$31.00 ©2023 IEEE

II. Related Work

Monte Carlo tree search (MCTS) is an algorithm that has been used to great success in fully-observable games (Gelly et al. [1]). It builds a search tree of alternating state and action nodes, each of which contains a visit count N and an estimated value V (initially both set to 0). Repeated simulations are performed from the root of the search tree, and one node is added to the tree each time. A rollout is done from each added node until a terminal state; then, the estimated value of the node is backpropagated up the tree. This process is continued until time is up or a certain number of simulations have been run; then, the action with the highest value or visit count is taken. The UCT policy (Gelly et al. [1]) selects which action to take during simulations and incorporates the visit count to add an exploration bonus (weighted by a parameter C) to nodes that have not been explored much.

Hearthstone is a popular CCG that is similar to Duelyst, with the greatest difference being that it does not have a grid board. Dockhorn et al. [2] use a modified version of MCTS that searches three plies (a *ply* is one turn for one player) into the future to play *Hearthstone*. The rollouts for this MCTS variant greedily select actions using a static evaluator and stop at the end of the ply, instead of the end of the game. When an initial run of MCTS for the agent's ply is complete, the n best end-of-turn states reached during rollouts are used as root nodes for MCTS searches for the opponent's next ply, and this is repeated to simulate the agent's next ply after that. The evaluations of the final states (three plies ahead) are backpropagated to the current-ply search tree. The algorithm returns the best sequence of actions from the search tree (until the frontier) instead of just the best next action to take. The opponent's deck and hand are predicted based on the opponent's previous plays.

Magic: The Gathering (MTG) is a popular trading card game that is a more complex version of *Hearthstone*. Cowling et al. [3] implement an expert-rule-based player for their own version of MTG that is simpler than Duelyst. Cowling et al. [3] use an ensemble of determinized binary MCTS trees whose final action values are summed to consistently beat the expert-rule-based player.

Partially observable Monte-Carlo planning (POMCP) is an algorithm invented by Silver and Veness [4] to apply MCTS in partially-observable domains. The search tree alternates between observation and action nodes. Observation nodes contain unweighted particle belief states that are updated during simulations. A simulation starts by sampling a state from the root's belief state so as to continue simulating from a fully observable state.

Settlers of Catan (SoC) is a complex 4-player POSG. Dobre and Lascarides [5] use POMCP to play SoC. They cluster actions into types — such as road building, city building, and trading — and learn from game data a preference distribution over those types. This distribution is used to influence which actions get selected during tree search and rollouts (an action type is sampled first, then an action from that type) (Dobre and Lascarides [5]).

III. Duelyst

In a Duelyst match, two players battle with 39-to-54-card decks on a 9x5 grid board. A match starts with a mulligan phase where both players simultaneously choose which cards to replace from their starting hands. There is a maximum hand size of 6. Each player controls a General, and the goal is to kill the opponent's General (by reducing that General's Health to 0). To do this, a player can play minion, spell, and artifact cards from their hand by spending mana. Minions are played on the board and can move and attack enemy units (enemy units are the opponent's minions and General). Spells have effects when played. Artifacts boost the power of a player's General. Duelyst is partially-observable because each player cannot see their opponent's deck or hand.

Duelyst has a discrete but large state space. There are 6 factions, each with 34 cards, and 112 neutral cards. A player's deck can be made up of cards from one faction and neutral cards and can contain up to 3 copies of the same card, which means there are on the order of $\binom{(34+112)\times 3}{54} \approx 10^{69}$ possible decks for a given faction. 45 units can be on the board at once, 6 cards in each player's hand, and potentially more than 54 cards in each player's deck (because some cards add cards to a player's deck). Units on the board have integer Attack, ranging from 0 to 99, and integer Health, ranging from 1 to 99. Units can also have an unlimited number of additional abilities granted to them by cards. All of this makes the state space enormous, not to mention the number of possible worlds when taking partial observability into account.

Duelyst has a discrete action space that can be either relatively small or very large, depending on how many cards are in the player's hand and how many units are on the board. It is not uncommon to have a branching factor of over 100 at some points during the game. In addition, several actions can be taken in a turn (depending on a player's mana, cards in hand, and units on the board) within a 90-second time limit and some actions have stochastic outcomes.

IV. The Starter AI

The starter AI is an expert-rule-based player. This means that it follows a set of hard-coded instructions about how to play the game, instructions which were manually created using expert-level human domain knowledge. These instructions lead to rigid, predictable behavior. For instance, the starter AI will always replace a card from its hand as its first action even though it may be beneficial in some cases to wait or not replace a card at all. To be able to handle the large state space of Duelyst, most of the instructions are somewhat general (e.g. play the most expensive card in hand first). However, the starter AI does rely on *intents* — rules for how to play specific cards effectively. The AI has intents for most of the cards in the starter decks (the decks players first unlock).

A proficient human player can consistently beat the starter AI with no trouble. Currently, the AI is only used for training new players. An interactive demo of a human (called You) playing — and winning — against the starter AI (called Argeon Highmayne) can be found at https://tinyurl.com/d2replay.

V. One-ply MCTS for CCGs

Our algorithm for Duelyst is called *one-ply MCTS for CCGs* (1MC). It is based on MCTS but incorporates four enhancements:

1) Rollouts stop at the end of the turn, instead of the end of the game, because the opponent's turn cannot be accurately simulated without a good model of their deck and hand. Rollouts either take random actions or do hill-climbing, greedily choosing the action that leads to the highest-valued state (according to the static evaluator). The latter is the approach taken by Dockhorn et al. [2].
2) Inspired by Dobre and Lascarides [5]'s use of action types to guide search, action types are an explicit part of the MCTS search tree — the possible actions at a particular state are grouped into an *action tree* (see Fig. 1). Duelyst's action space can be neatly factored into seven action types: Mulligan, MoveUnit, AttackWithUnit, PlayCard, ReplaceCard, EndTurn, and FollowUp (a FollowUp action can occur after a PlayCard action or another FollowUp action). The MoveUnit, AttackWithUnit, and PlayCard action types can be further factored (in the next layer of the action tree) into *action sources*: the cards that can perform that type of action (e.g. MoveUnit could branch into UnitA and UnitB). Beneath action types or action sources are ground actions, which are fully specified actions that can be executed in the current game state (e.g. where to move UnitB). The EndTurn action type is unique in also being a ground action. Fig. 1 shows an example of an action tree for a particular state. Action types are blue, action sources are purple, and ground actions are green. Each action type and action source node has a visit count and stored value, just like the other nodes in the search tree, and choosing an action consists of choosing an action type, potentially an action source, and then a ground action.
3) Mulligan actions are simultaneous, so, as part of the state transition after a simulated mulligan, the agent pretends that the opponent takes an arbitrary Mulligan action. If the agent is going second, it will also ignore

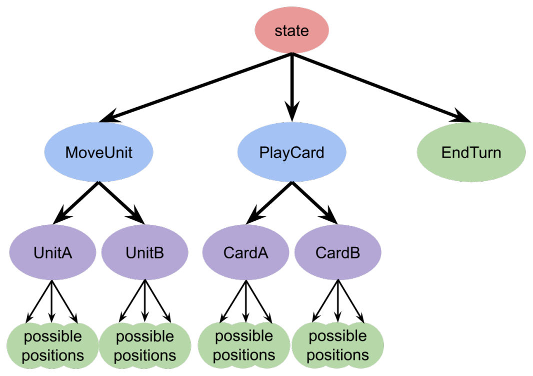

Fig. 1. An action tree for a particular state.

the opponent's first turn (pretend that the opponent only takes the EndTurn action) in order to continue simulating in the agent's first turn.

4) If there is only one action available to the agent, that action is immediately chosen instead of doing a tree search for the allotted time.

The parameters for 1MC are: the rollout policy to use (random or greedy), the time to spend searching for each action, and whether or not to use action trees, as well as the default MCTS exploration parameter C.

VI. Static Evaluator

The static evaluator we created estimates the value of a state using a set of simple rules. A win state has a value of 10,000 multiplied by the remaining health of the winner (in order to differentiate end-game states). A lose state is the same but with a negative value. Any other state's value is the difference in values of both players' units. A unit's value is based on its cost, Attack, and Health, and there is a penalty for being far from mana orbs (collectible resources that start on the board at the beginning of the game) and a bonus for being near enemy units (especially the General) that it can kill. A General's Health is valued at three times that of a minion to promote the ultimate objectives of staying alive and killing the enemy General.

VII. Experimental Results

We conducted two experiments in Duelyst. The first was to tune 1MC's parameters and the second was to compare the performance of 1MC to that of baseline algorithms. For both experiments, both AI players use the first deck players unlock — the Lyonar starter deck. This is so neither player has a better deck than the other and so the starter AI can effectively use its cards. Unlike in a true Duelyst match, the players are given unlimited time to take their turns.

A. *Tuning 1MC*

In order to see the effect of action trees, random or greedy rollouts, and choosing actions by value or visits on the performance of 1MC against the starter AI, we ran 1,000 games with 1MC using each combination of possible arguments. In every case, 1MC had 15 seconds to choose each action and C was set to 10.

The average winrate for each combination is shown in Table I. With random rollouts, choosing by value is slightly better than choosing by visits, and use of action trees does not seem to make a difference. With greedy rollouts, choosing by value is significantly better than choosing by visits, and action trees only make a difference when choosing by visits — but it is a significant difference. When choosing by value, both random and greedy rollouts perform about the same. This is likely due to the fact that random rollouts are faster while greedy rollouts are more accurate, which leads to the same quality of gathered information in a fixed amount of time. This speed-accuracy trade-off could also explain why action trees only lead to a higher winrate when choosing by visits and using greedy rollouts — since fewer greedy rollouts occur than random rollouts, the visit counts of actions may not vary much even though the values do, but when actions are factored into trees, there are fewer top-level choices and thus the visit counts vary more.

Seven of the eight argument combinations allow 1MC to beat the starter AI consistently (a greater than 50% winrate). Six of those seven successful methods achieve above a 60% winrate, with the highest being 65%. The two weakest combinations use greedy rollouts and choose actions by visit count.

B. *Comparative Performance*

In order to see if a MCTS-based approach was necessary to consistently beat the starter AI, we created the following baseline algorithms: **Random** takes a random action. **Hillclimb** simulates trying each action and chooses the one that leads to the highest-valued state (1-step lookahead). **Rollout** simulates trying each action, performs n rollouts (random or greedy) from the resulting state, and chooses the action that leads to the highest-valued end-of-turn state.

We ran Random, Hillclimb, Rollout with random (n = 1 and 7) and greedy (n = 1) rollouts, and 1MC with random and greedy rollouts against the starter AI 1,000 times each. Rollout (7 Random) was chosen because a preliminary experiment showed that increasing the number of rollouts improved the algorithm, but going beyond 7 caused it take too long to choose actions. Both 1MC versions had 15 seconds to choose each action, chose actions by value, and used a C of 10. 1MC (Random) did not use action trees, but 1MC (Greedy) did.

TABLE I
1MC Tuning

rollouts	choose by	action tree	winrate (%)
random	visits	no	62
		yes	61
	value	no	63
		yes	64
greedy	visits	no	34
		yes	51
	value	no	65
		yes	64

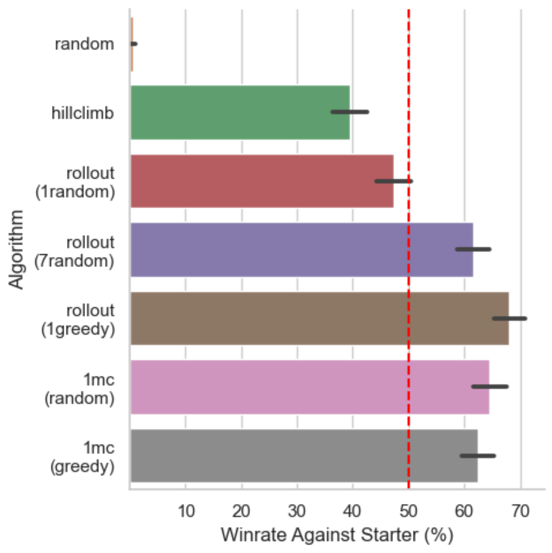

Fig. 2. The winrate of each algorithm against the starter AI.

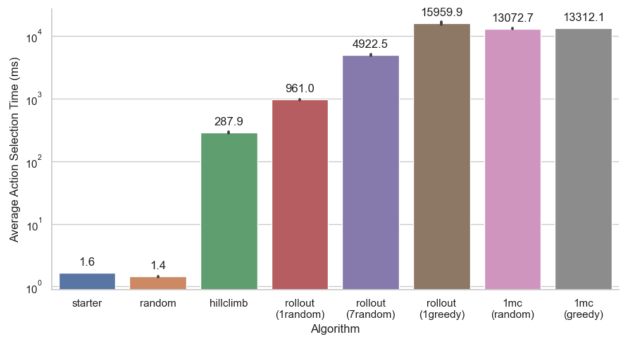

Fig. 3. The average time (ms) that each algorithm takes to select an action.

The winrate of each algorithm is shown in Fig. 2. Random, Hillclimb, and Rollout (1 Random) are not able to beat the starter AI consistently, but, as expected, Rollout (1 Random) outperforms Hillclimb, which outperforms Random (by a large margin). Both 1MC versions have similar winrates over 60%, as in the last experiment. What is unexpected is that Rollout (7 Random) has a winrate in the same range as 1MC and Rollout (1 Greedy) outperforms 1MC (although not significantly), with a 68% winrate. This could be because C is too high, causing 1MC to explore bad actions a bit too much, while Rollout explores each action an equal amount.

Fig. 3 shows the average selection time of each algorithm, including the starter AI. Random is naturally the fastest, with the starter AI a close second. Rollout (1 Greedy) is the slowest, with an average of 16 seconds per action. Both 1MC methods take around 13 seconds on average. These successful approaches are too slow against a human, but Rollout (7 Random) only takes 5 seconds per action, which is reasonable.

VIII. Discussion

The results show that a general planner even simpler than MCTS can consistently beat the specialized starter AI and take actions in a reasonable time.

We worked in the *Duelyst II* codebase, which is written in JavaScript and has limitations. Originally, there was no equality comparator for game states, which is necessary for MCTS, so we had to implement one. The biggest problem is that the existing state copying function serializes the large game state object to a JSON and then deserializes it, which is incredibly slow. We plan to implement a more efficient method of state copying that avoids copying information that is not relevant to simulations. Faster state copying would speed up most of the algorithms, which is important for having the most successful ones be viable options in the actual game (players do not want to wait 16 seconds for the AI to make a move).

Another limitation of this work is that we have not yet tested different values of C or different time limits for 1MC. It would be interesting to see what the cut-off in performance is and if planning for less time can still yield intelligent actions.

A possible extension that we have already started experimenting with is multi-ply planners: algorithms that maintain a belief about the opponent's deck and hand and simulate the opponent's turn. Preliminary results in this direction are unpromising, possibly because the static evaluator needs to be improved in order to accurately look further ahead.

IX. Conclusion

We introduced a MCTS variant, one-ply MCTS for CCGs (1MC), and the notion of factoring actions into action trees based on their types and sources. We pitted 1MC and some baseline algorithms against the expert-rule-based starter AI and found that not only 1MC but also two of the baselines could consistently win. General-purpose algorithms can be superior to specialized ones, even in complex POSGs.

Acknowledgment

This research was supported in part by NSF grant 2008594.

References

[1] S. Gelly, L. Kocsis, M. Schoenauer, M. Sebag, D. Silver, C. Szepesvári, and O. Teytaud, "The grand challenge of computer go: Monte Carlo tree search and extensions," *Communications of the ACM*, 2012.

[2] A. Dockhorn, M. Frick, Ü. Akkaya, and R. Kruse, "Predicting opponent moves for improving Hearthstone AI," in *Information Processing and Management of Uncertainty in Knowledge-Based Systems*, 2018.

[3] P. I. Cowling, C. D. Ward, and E. J. Powley, "Ensemble determinization in Monte Carlo tree search for the imperfect information card game Magic: The Gathering," *IEEE Transa. Comp. Intell. and AI in Games*, 2012.

[4] D. Silver and J. Veness, "Monte-Carlo planning in large POMDPs," in *Proceedings of NIPS-10*, 2010.

[5] M. Dobre and A. Lascarides, "POMCP with human preferences in Settlers of Catan," in *Proceedings of AIIDE-18*, 2018.