Interactive visualization system of adversarial search algorithms

Ma, Xueran/Sherry

December 2020

1 Abstract

AI games usually have too many algorithms behind it and its "thinking" process is complex to get a concrete idea for players. Although there are many good adversarial search algorithms right now, it is still too abstract to connect those complex algorithms with the games we are playing together, also it is necessary to clarify how the game tree expand while keeping the environment as invariant. In this work, I will design an AI game tree visualization system which enable flexible visual adversarial search algorithm exploration by allowing the user to modify parameters of AI algorithm – Monte Carlo Tree Search, playing against the AI and see the real-time animation. During the playing procedures, for each predicted step of AI, the system will visualize the game tree and the animation of its expanding and the corresponding properties of the AI algorithm. The system will deploy a simple game engine for a simple game, connect-4, and synchronize another interface for the game tree algorithm interactive animation. With the system, the user can explore to find better properties (for example iteration) of AI algorithm, also could find a better combination of parameters. On the other hand, for beginners, the system could work perfectly as a visualization tool to comprehensive the complex AI algorithm quickly. In this paper, I will introduce how to implement the system in details, including three parts: game engine implementation, game tree algorithms, interactive and real-time interactive animation and visualization.

2 Motivation

As adversarial search is becoming more popular, AI agents are part of our entertainment or business life. People are more and more curious about why AI could win them in chess and other games especially children. Learning how AI "think" using the system I proposed is better and easier to form logical thinking for children and get concrete concepts of adversarial search algorithms for curious people such as Monte Carlo Tree Search. Because of its one of properties – confrontation, this interactive visualization system is also a good educational tool to arouse people's enthusiasm to learn the adversarial algorithm. In other words, people learn through entertainment.

3 Related Work

In terms of adversarial search algorithms or game tree search algorithms, there are many categories and classic algorithms right now. What's important is to pick classic ones for each categories to implement and compare or choose an advanced one with enhancement on classic algorithm. Several classic and basic algorithm are MiniMax, Alpha-Beta pruning, depth-limit search, iterative deepening and so on. Among those, Monte Carlo Tree Search has enhancement based on MiniMax and is also a more complex game tree structure with several stages. Although there are many academic researches and theories on the adversarial algorithms, applications to visualized or animated these theories are still valuable and their practical significance are overlooked. [1]

Connect-4 is a finite two person zero-sum sequential game, indicating that two actors being involved make their moves in sequence (take alternating turns to drop pieces into the slots and pieces fall to the bottom of the board) and have the opposite goal, thus the sum of gain for all players equals zero at the end. To win, player needs to get 4 pieces in a row, vertically, horizontally or diagonally.

Traditionally, in a phd level, a game could be defined by a complex tuple. However, it is almost impossible to work with and thus a well-known data structure – game tree was proposed to represent a game. Several properties of game tree includes: state of game – initial node, terminal node and intermediate node, move/action, branching factor. For such adversarial game like alpha-go, different algorithms try to find the most promising next move.

Some classic algorithms like MiniMax algorithm, Alpha-Beta algorithm are applied in many AI game like alpha-go. They make AI games magical and secrete. I gonna uncover the secrete by visualizing the game tree and even provide the "controller" to the user so that they could combine different strategies and create their AI agent in connect-4.

4 Methods

In this system, I animated the real-time process on the Monte Carlo Tree Search while user plays again the AI. In this section, I will list

1. Some intuitive descriptions on the baseline algorithm of Monte Carlo – MiniMax and Monte Carlo itself.

2. Some other strategies to improve the traditional algorithms

3. A description about the structure and process of the implementation of the system. Details about system results will be introduced with graph in section Experimental Results.

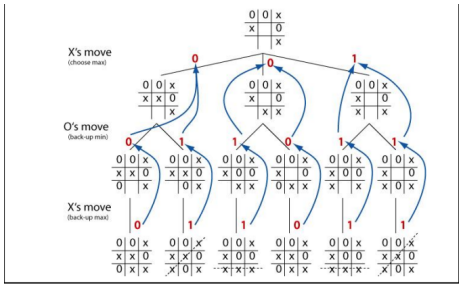

Figure 1: Minimax game tree sample of Tik-Tok game [1]

4.1 MiniMax

Definition 4.1 (MiniMax)**.** Maximizes the payoff given the other player plays the best possible game X.

$$v_A(s_i) = \max_{a_i} v_B(move(s_i, a_i)) \quad v_A(\hat{s}) = eval(\hat{s}) \tag{1}$$

$$v_B(s_i) = \min_{a_i} v_A(move(s_i, a_i)) \quad v_B(\hat{s}) = -eval(\hat{s}) \tag{2}$$

vA and vB are utility functions for players A and B respectively (utility = gain, payoff or reward). move is a function that produces the next game state (one of the current node children) given current state si and action at that state ai. eval is a function that evaluates the final game state (at terminal node) and s is any final game state (a terminal node) minus sign at vB for terminal state is to indicate that game is a zero-sum game. For the classic algorithm, it has many weakness. It has to traversal the whole tree which is expensive. For the following algorithms and strategies, they could improve and avoid the weakness in different aspects.

4.2 Monte Carlo Tree Search

Just like Figure 3, the algorithm involve 4 stages. Initially the tree hasn't been explored, and node being not-fully expanded means that there exists unvisited child. The game tree is expanded using rollout policy function(3) which consumes a game state and produces the next move. In many cases, it is usually uniform random function, that said, pick unvisited node randomly. The rollout policy have many selections, probably could be considered as a dynamic properties in out system. The final step is to traversal back from the leaf node (where simulation started) up to the root node and update the statistic for each node.

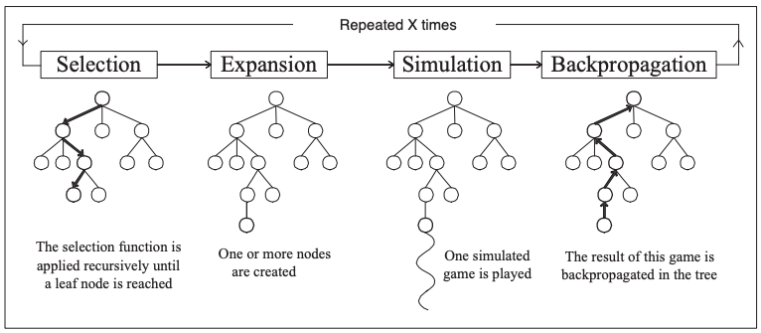

Figure 2: Outline of a Monte-Carlo Tree Search [2]

Definition 4.2 (RolloutPolicy)**.**

$$s_i \rightarrow a_i \tag{3}$$

After the tree is fully expanded, it will repeat the whole process but then it will use the UCT function to compute the node statistic and decide which move to take.

Definition 4.3 (UCT)**.** Upper Confidence Bound applied to trees, used to choose the next node among visited nodes to traverse through.

$$\cup CT\left(v_i, v\right) = \frac{Q\left(v_i\right)}{N\left(v_i\right)} + c\sqrt{\frac{\log(N(v))}{N\left(v_i\right)}} \tag{4}$$

The formula is constructed by two parts: the first division is exploitation/winning rate – Q(v) is the total simulation reward/gain/utility, and N(v) is the total number of visits for the specific node. Basically, exploitation is the socre to decide how promising the node is and how intensively explored it has been. The second part in the formula is exploration component, it exists because we are afraid that the search will end up greedily exploring only winning nodes very early of the search. So the c will control the tradeoff between the two parts.

4.3 Heuristic and policy functions

A heuristic function is a function which ranks alternatives in search algorithms at each branching step based on available information to decide which branch to follow. And a policy function defines action realted to a state. They could be estimation strategies of game AI and actually so many selections to choose. Some possible heuristic or policy functions: stochastic policy, deterministic policy (greedy), Killer Move Heuristic Some other strategies: iterative deepening, depth-limit

How the system is constructed and how each stage of Monte Carlo search Algorithm is visualized and animated in our system.

4.4 Implementation process and description

The system is mainly developed using graphviz for graph visualization and position generating, networkx for complex network structures construction, pygraphviz for integrating python and graphviz, matlibplot animation for interactive animation and tkinter for interactive game and canvas. The whole system consists three parts: game UI and engine, MTCS tree structure for

MTCS implementation, and MTCS GUI for animation and visualization. Monte Carlo Tree Search is built by Node structure. From the root of the current game state, tree policy function is called to decide if it goes into an expanding stage or selection stage by judging if the current node is fully explored. Expanding stage will call the rollout function – randomly choose an unvisited child to visit and thus add one node to the graph. Selection stage will caculate the UTC score for all the visited children and select the one with best score to call the tree policy again until an unvisited child is found. After an unvisited child is found, it enters the simulation stage, which will randomly select next move until one wins or lose or draw where it caculates the reward for this simulation as 1, -1 or 0. Each step of simulation will continue add to the graph to visualize until it enters the backpropagation stage. The backpropagation stage will back up the reward of this simulation to all the parents and thus in our visualization, the simulation nodes disappear and instead the visit times and reward value of their parents are updated. The whole process will repeat as the user wishes as the system allows users to modify some parameters in the main canvas.

5 Experimental Results

5.1 Connect-4 main Game

As figure 4, it is the main UI for the Connect 4 Game. It has functions as followings: two modes for tradeof between better visualization and smarter AI – one is play mode, it doesn't have realtime visualization function because its property of high iteration makes the animation could last over hours to finish but it is far more smarter, another is the tutorial mode, it has a good animation for the game tree expanding while its iteration could only be set up to 30 comparing play mode's 10,000. What's more, the user could choose to set up the factor element of UTC function to see the difference of game tree.

5.2 Animation of Game tree

As figure 4, the root original has 7 unvisited child and thus it is not fully explored and randomly choose one child with visit times as 0. And another situation to visit is selection stage as figure 6, the children of the root (the

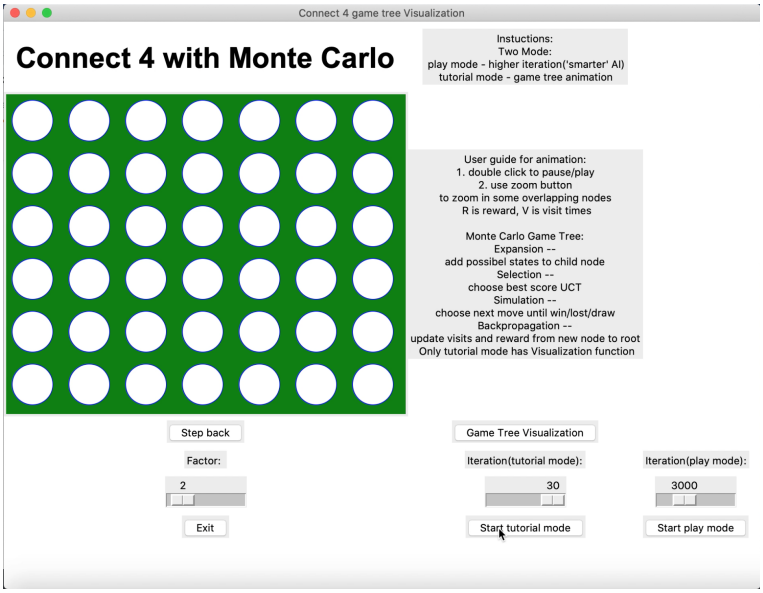

Figure 3: Main Game Board

current node) are all visited (V is at least 1 in the graph) so use UTC to caculate scores of children and select the best one to continue finding unvisited node. Once one unvisited node is found, as the fiture 7, simulation stage use blue nodes to simulate the moves step by step until one side win or lose or draw(as our figure 7 case, R is -1 cause human/yellow win). Although there are overlapping in the picture, in real system, the animation could be stopped by double click and use zoom button to zoom in. Once the simulation ends, all the blue nodes won't be kept inside the tree, comparing to figure 4, figure 5 updates all the parents of the visited node and itself by +1 for visits time and turn (human -1 and AI 1) * reward. The process will repeat many times and finally the tree will grow enough huge.

6 Summary

The purpose of this system was to make complex adversarial algorithm and AI grounded. Animation displays the real-time process of MTCS. However, the system still lacks of generalization and comparisons for other adversarial algorithms. This could be a significant improvement if achieved in the future.

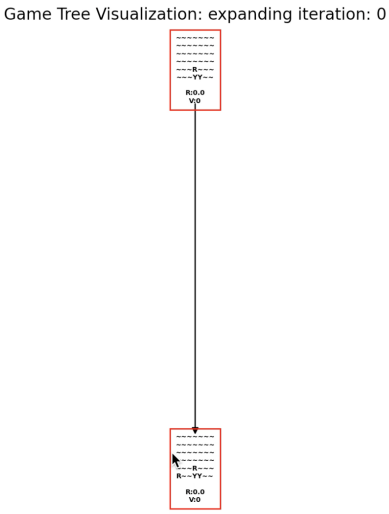

Figure 4: Expanding Stage

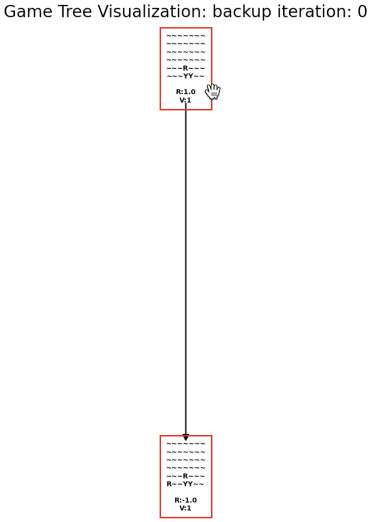

Figure 5: Backpropagation Stage

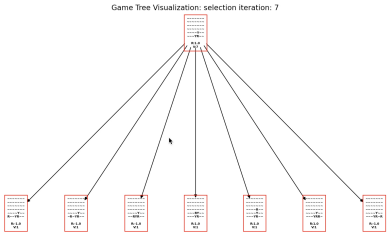

Figure 6: Selection Stage

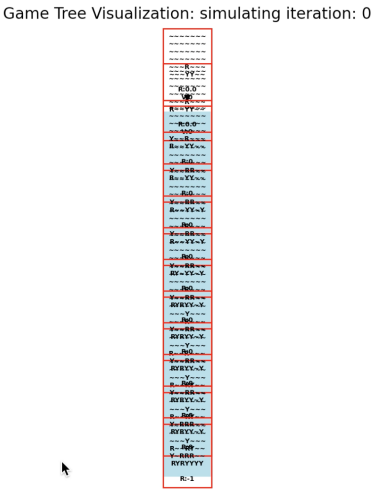

Figure 7: Simulation Stage

References

[1] Mostafa Abdel Aziem Mostafa Hesham Eldeeb Ahmed Elnaggar, Mahmoud E Gadallah. A comparative study of game tree searching methods. 2014.

[2] Istvan Szita Guillaume Chaslot, Sander Bakkes and Pieter Spronck. Monte-carlo tree search: A new framework for game ai.