Five In A Row Game

--An AI Player

Yingying Xia
Spring 2017
Prof. Craig Graci

http://cs.oswego.edu/~yxia/coursework/csc466/project/index.html
Abstract

*Five In A Row* is an abstract strategy board game often played with Go board and pieces. This paper write-up will introduce the implementation of an AI player with a collection of pattern detection, a collection of pattern detection and minimax algorithm. Furthermore, the goals, current accomplishments, successful demos, possible improvements and future plans of the project will also be introduced.

Introduction

The project I have chosen to complete throughout my Spring 2017 semester is to create an AI player for a board game called *Five In A Row*. This program is implemented completely from scratch in the programming language of LISP. The program currently could be played against the machine itself and against a human player. The ultimate goal of this project is to have the AI player make reasonable decisions when played against human players within an efficient amount of time.

Background

I have chosen to implement this board game mainly because it was a significant part of my childhood. In addition, I wanted to utilize this opportunity to be more familiar with the usage of Minimax Algorithm as well as to investigate more about LISP’s syntax and unique features. *Five In A Row* is a name I translated directly from Chinese *(五子棋) qí* for the board game. In English, the board game is usually called *Gomoku* or *GoBang*. It is a two-player board game often played on a Go board and pieces. It is more often seen in sizes of 15 by 15 or 19 by 19. The rule of this game is simple and is similar to it of Tic-Tac-Toe; the player who first obtains a consecutive five pieces on
the board is the winner. However, *Five In A Row* is considerably more difficult than Tic-Tac-Toe because of its huge searching space, and needs better pattern constructing strategies.

The efficiency of the strategies played by a programmed AI is dominated by how well the evaluation function and the searching algorithm makes decision. A threat is a very important threat notion in the game. Those threat needs to be terminated on time or it will result in a defeat very easily. Since *Five In A Row* is one of those logic games and is played with open rules, which means both side of players know the available moves from a given point in the entire game, a look-ahead algorithm will allow the machine player to play smarter by doing some heavy space searching. According to the a previous paper, both human players and AI machine spend time to look for threat sequences; “human use methods of optimistic search and verification” while “programs use conventional tree-search techniques” [4]. Because computer does not have the natural thinking ability like humans do, it has to do some extra searching with evaluation to make up this shortcoming. One of those look-ahead, search-tree structured algorithm is the Minimax algorithm. The Minimax algorithm is often accompanied by Alpha-Beta Pruning, which is a heuristic that furthermore optimizes the searching space, in many of the previously done researches and projects. They have also been proven efficient and successful in two-player games.

Program Description

I. **Board Representation**

Searching and look-ahead algorithms like Minimax algorithm require data structures like search trees, where nodes represents states and edges represents operators that leads from one node to another. In this case, it would be beneficial to represent game board information as an entity of node
class, and the move that updates the game state to the next as an edge. The current game state consists all existing pieces is simply represented by a list in LISP. In addition, a node in this project will also keep track of the parent node, the previous operator that leads to the current node, the next available moves, evaluation score, and the move history. All information about a node will be updated as it traverses through the entire search tree.

II. Pattern/Threat Recognition

The implementation of this game is based on translation of rows, columns and diagonals into sequences of strings in lists of 5 lengths and 6 lengths; because some patterns require checking 5 consecutive positions to determine whether patterns exist while others require checking 6 \[5\]. The translation of sequences occurs four directions for each position on the board – downward vertically, rightward horizontally, left downward diagonally and right downward diagonally (see figure 1, shown in sequence of length 5 only). Both players, essentially, need to be aware of some threat patterns their opponent constructed, but also to construct threat patterns for their opponent. Threat patterns refer to those patterns, in which if they are not blocked on time, will result in a loss very soon. The patterns implemented in the program, from figure 1 to figure 3 in order of left to right and top to bottom, are open three, capped three, consecutive five, open four, capped four, gapped three, gapped four, and gapped two-two. Open three, capped three, consecutive five, gapped four, gapped three and gapped two-two require a 5-length sequence check, while the rest requires a 6–length sequence check.
III. Evaluation Function

The evaluation function of the program returns the measure of a current game state in the numerical form. For each game board, the evaluation function calculates the number of each pattern exist on the entire board multiply by the evaluation score given to each pattern, for all patterns. In addition, a distance calculation of all pieces from the center (position J J) is subtracted from a constant. Therefore, the closer a piece is placed from the center, the shorter distance. With a shorter value of distance subtracted from a constant, the larger difference. This distance calculation is mainly used to hold pieces together. When collecting the number of each pattern, the program limits itself to spaces that are neighbors of all existing pieces. Because a game board most of the time remains empty, it is meaningless to search through all possible spaces on the board. An example of the checking space is shown in figure 4; instead of looking through the entire board, which mostly contains empty spots, the program only searches through the spaces that are within the red line. Evaluation score given to
each pattern depends on the distribution of that pattern leading to a possible win; for example, constructing an open four is obviously better than constructing an open three, and constructing an open three is better than constructing a capped three. The score assigned for each pattern is as follows:

<table>
<thead>
<tr>
<th>Pattern Name</th>
<th>Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>Open Three</td>
<td>100000</td>
</tr>
<tr>
<td>Capped Three</td>
<td>10000</td>
</tr>
<tr>
<td>Consecutive Five</td>
<td>1000000</td>
</tr>
<tr>
<td>Open Four</td>
<td>1000000</td>
</tr>
<tr>
<td>Capped Four</td>
<td>100050</td>
</tr>
<tr>
<td>Gapped Three</td>
<td>100000</td>
</tr>
<tr>
<td>Gapped Four</td>
<td>100050</td>
</tr>
<tr>
<td>Gapped Two-Two</td>
<td>100050</td>
</tr>
</tbody>
</table>

For example, if a game board contains one open three and two capped fours for the player who holds the piece X, the evaluation score evaluated by the function will be: 1*100000 + 2*100050 + (constant – total distance of all pieces from center).

IV. Minimax Algorithm

The Minimax Algorithm generates the search tree involving two players, MAX and MIN. The search tree is generated depth-first starting at the root up to a given depth limit (in my program the depth limit is 2)\(^7\). The last layer of game states, called the leaf nodes, is then evaluated using the evaluation function. After the evaluations, the nodes that belong to the MAX player will receive the board with the maximum board score of its children while the nodes that does belong to the MIN player, MAX will receive the board with the minimum board score of its children\(^7\). The minimax algorithm predicts moves ahead for the MAX player assuming the MIN player will result in a not-so-good game state for MAX. Overall, the minimax algorithm allows the board evaluation
function determine how good or how it is for a move to be performed in the perspective of the MAX player, by constructing a tree-like search structure. The Minimax algorithm in the program also takes in a parameter of current-player and thus produces four possible results. For the current player who is not MAX, the algorithm tends to return the board with the minimum score for MAX. For the current player who is the MAX, the algorithm tends to return the board with the maximum for MAX.

<table>
<thead>
<tr>
<th></th>
<th>Current player is X</th>
<th>Current player is O</th>
</tr>
</thead>
<tbody>
<tr>
<td>MAX is X</td>
<td>X’s best</td>
<td>X’s worst</td>
</tr>
<tr>
<td>MAX is O</td>
<td>O’s worst</td>
<td>O’s best</td>
</tr>
</tbody>
</table>

V. Strategy Heuristic

Through running some trials of both machine-to-machine and machine-to-human demos, I came to realize that the best move (or the best next board) produced by minimaxing for one player does not necessarily mean it is the worst move for the opponent. Since the evaluation function only measure a game state for one player’s perspective, it does not take into account how well the game state currently is for the opponent. Thus, the program added another layer of heuristics on top of the Minimax algorithm, for the purpose of teaching the program when to defend itself by blocking the opponent from proceeding threat patterns, and when to construct threat patterns to benefit itself. For example, in figure 6, player SAM made a move at position (L,K), which created an open three that needs to be blocked right away.

![Figure 6](image1)

![Figure 7](image2)
However, as shown in figure 7, the minimax algorithm produced the best move for player TEEMO is to put its piece at position (K,L) instead of blocking SAM’s open three, because the evaluation function have also detected an open three for TEEMO. This is obviously not the best move for the game even though it has the best score; SAM will be the no-doubt winner by placing its piece next to the open three to form an open four, which is impossible to be blocked. The strategy heuristic in the program allows the current player to decide whether to call for its best or call for opponent’s worst. If the opponent has no threat patterns that need to be blocked, the current player would choose to construct its own threat pattern for the opponent; if the opponent has threat patterns that need to be terminated, the current player would choose to defend itself by calling for the opponent’s worst. Of course, if a greater depth for Minimaxing is provided, this heuristic could have been avoided, but the tradeoff would be more memory consumption and longer processing time. In this project, the searching depth in minimaxing is 2 so it is quite important to have this heuristic for the AI to make reasonable moves.

Demos

I. Board Representation – The Node Class

[2]> (setf n (new-node))  //creating a new node/empty node
#<NODE #x1ABADAD5>

[3]> (setf n (apply-move (cons 'j 'j) n 'x))  //put an X at position J,J
#<NODE #x1ABAE1E5>

[4]> (info n)  //visualize game board and print out other information

===================================INFO===============================
Node:#<NODE #x1ABAE1E5>
Current Board:
A B C D E F G H I J K L M N O P Q R S

- - - - - - - - - - - - - - - - - - A
- - - - - - - - - - - - - - - - - - B
- - - - - - - - - - - - - - - - - - C
- - - - - - - - - - - - - - - - - - D
Current Board:

```
   - - - - - - - - - - - - - E
   - - - - - - - - - - - - - F
   - - - - - - - - - - - - - G
   - - - - - - - - - - - - - H
   - - - - - - - - - - - - - I
   - - - - - - - - - - - - - X
   - - - - - - - - - - - - - J
   - - - - - - - - - - - - - K
   - - - - - - - - - - - - - L
   - - - - - - - - - - - - - M
   - - - - - - - - - - - - - N
   - - - - - - - - - - - - - O
   - - - - - - - - - - - - - P
   - - - - - - - - - - - - - Q
   - - - - - - - - - - - - - R
   - - - - - - - - - - - - - S
```

Length of the available moves for this board is: 360

Parent Board:

```
#<NODE #x1ABACD1D>
```

Node Derived from parent with the move: (J . J)

Board Eval Score in X perspective: 2621.2173
Board Eval Score in O perspective: 2621.2173

Move history: ((J . J))

```
NIL
```

II. Pattern Recognition

```
[12]> (info n) ;;a board with existing pieces
```

```
NIL
```

```
```

Node: #<NODE #x1ABB1975>

Current Board:

```
ABCD EFGHIJKLMNOPQRSTUVWXYZ
```

```
   - - - - - - - - - - - - - A
   - - - - - - - - - - - - - B
   - - - - - - - - - - - - - C
   - - - - - - - - - - - - - D
   - - - - - - - - - - - - - E
   - - - - - - - - - - - - - F
   - - - - - - - - - - - - - G
   - - - - - - - - - - - - - H
   - - - - - - - - - - - - - O
   - - - - - - - - - - - - - I
   - - - - - - - - - - - - - J
   - - - - - - - - - - - - - K
   - - - - - - - - - - - - - L
   - - - - - - - - - - - - - M
   - - - - - - - - - - - - - N
   - - - - - - - - - - - - - O
   - - - - - - - - - - - - - P
   - - - - - - - - - - - - - Q
   - - - - - - - - - - - - - R
   - - - - - - - - - - - - - S
```
NIL
Length of the available moves for this board is: 354
Parent Board:
#<NODE #x1ABB1619>
Node Derived from parent with the move: (K . J)
Board Eval Score in X perspective :2618.803
Board Eval Score in O perspective :102616.39
==============================================================END INFO==============================================================
NIL
[13]> (getindex 'h 'k)
143
[14]> (get-horizontal n 143)
(- - - - -)
[15]> (get-vertical n 143)
(- O O O -)
[16]> (setf verticallist (get-vertical n 143))
(- O O O -)
[17]> (openthree verticallist 143 'o)
T

III. Evaluation Function
[30]> (display-n n) //displaying the board

A B C D E F G H I J K L M N O P Q R S

- - - - - - - - - - - - - - - - - - A
- - - - - - - - - - - - - - - - - - B
- - - - - - - - - - - - - - - - - - C
- - - - - - - - - - - - - - - - - - D
- - - - - - - - - - - - - - - - - - E
- - - - - - - - - - - - - - - - - - F
- - - - - - - - - - - - - - - - - - G
- - - - - - - - - - - - - - - - - - H
- - - - - - - - - - - - - - - - - - I
- - - - - O X O - - - - - - J
(setf m (minimax n 2 'o 'o))  //looking two moves ahead for O’s best
NIL

[31]> (board-evaluation n 'x)  //board score in X’s perspective
202613.64
[32]> (board-evaluation n 'o)  //board score in O’s perspective
102616.39

IV. Minimax Algorithm

[33]> (display-n n)  //current board

| A | B | C | D | E | F | G | H | I | J | K | L | M | N | O | P | Q | R | S |
|   |   |   |   |   |   |   |   |   |   | X | O |   |   |   |   |   |   |   |
| X |   | X | O |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
|   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
|   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
|   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
|   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
|   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
|   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
|   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
|   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
|   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
|   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
|   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |

NIL

[34]> (display-n m)
V. Strategy Heuristic

// Sam is forming an open three
SAM is thinking.....might take awhile......
SAM made a move at position (L . K).

// Teemo chose to defend first even though he also can form an open three or capped four. Otherwise he will lose.

TEEMO is thinking......might take awhile......

A B C D E F G H I J K L M N O P Q R S

- - - - - - - - - - - - - - - - - - - - - - - A
- - - - - - - - - - - - - - - - - - - - - - - B
- - - - - - - - - - - - - - - - - - - - - - - C
- - - - - - - - - - - - - - - - - - - - - - - D
- - - - - - - - - - - - - - - - - - - - - - - E
- - - - - - - - - - - - - - - - - - - - - - - F
- - - - - - - - - - - - - - - - - - - - - - - G
- - - - - - - - - - - - X - - - - - - - - - - - H
- - - - - - - - - - - - X X O - - - - - - - - - I
- - - - - - - - - - - - O O X X X - - - - - - - J
- - - - - - - - - - - - X O O - - - - - - - - - K
- - - - - - - - - - - - O O - - - - - - - - - L
- - - - - - - - - - - - X - - - - - - - - - - M
- - - - - - - - - - - - - - - - - - - - - - - N
- - - - - - - - - - - - - - - - - - - - - - - O
- - - - - - - - - - - - - - - - - - - - - - - P
- - - - - - - - - - - - - - - - - - - - - - - Q
- - - - - - - - - - - - - - - - - - - - - - - R
- - - - - - - - - - - - - - - - - - - - - - - S

TEEMO made a move at position (M . L).

Reflection and Conclusions

I. Original Goals

My original plans for this project is to implement an AI player that makes reasonable decision on
taking moves using both Minimax algorithm and Alpha-Beta Pruning for quick performance. My original goal also included incorporating machine learning schemes into the game so that the AI could learn from how humans play through machine-to-human games. I didn’t want the AI to start learning from zero, but from a moderate player to a human-like expert. I thought incorporating machine learning into my program could also improve performance because it could make quick decisions based on its learned strategy from human players, and only makes its own decision through heavy searching when it hasn’t learned yet.

II. Accomplishments and Realization of the Plans

I think ¼ of my original goals have been accomplished. I have originally divided the project into several tasks. I have realized that task 3, which was scheduled to detect a winning state of a game board, was meaningless because my pattern recognition and board evaluation can be used to detect winning state. I have also realized that representing the board, as a list in a node, is definitely not the best way to keep track of the game state; I had to do a lot of index checking when translating sequences of list for pattern recognition. So far, in the project, I have made the program to perform AI-to-AI gameplays as well as AI-to-Human gameplays. Many demos indicate that my Minimax algorithm and the strategy heuristic work moderately well (disregarding the computing speed). Because of the limited schedule we were given in a semester, I did not get the chance to implement Alpha-Beta Pruning and machine learning, which would significantly help fasten the searching speed when a greater number of moves are performed.

III. Future Plan
My future plan of this project is to implement Alpha-Beta Pruning and integrate machine learning into the program for performance and strategy improvement purposes. I would like to collect statistics on the computing time difference with and without Alpha-Beta Pruning. I would like to configure a database of winning sequence of moves for AI to learn from in order to increase its intelligence.
Bibliography