CS 470: Project 2 - Connect 4
University of Idaho
Matthew Waltz
9th March 2018

Abstract

This project implements an alpha-beta pruned minmax algorithm to play connect 4 against a human opponent. Coded in the C language, it plays rather well and quickly and is able to find a winning move in most every case. It also makes sure to block any potential winning moves by the human player and takes the winning position before blocking if one exists. It also performs rudimentary shortest winning path finding by selecting the best route found during the alpha-beta search. One interesting result is that it favors diagonal wins, or that may just be my horrible ability to see what it is going to do in the future. All in all though, the algorithm works which was the best feeling.

Contents

1 Algorithm

The following section describes various parts of the overall algorithm.

1.1 Min-Max

The core algorithm was designed around minmax which is implemented with alpha beta constraints. It works by maximizing for the AI, and then minimizing for the human player recursively. It simulates dropping a piece into the column and evaluates the position, and then undos that drop and repeats. The depth of the search is limited to a depth of 9 ahead in order to keep a reasonable space and time constraint. The children of each node then are produced by each successive simulated drop, which are then evaluated. Once the root node is reached, the evaluation function takes over.

1.2 Alpha-Beta

As an addition to the minmax algorithm, alpha beta pruning is used to help reduce the time. This is done by taking the return of the minmax and using it the next runs, narrowing the search field based on the maximum for the AI and the minimum for the human player. The INT_MIN and INT_MAX limits are used as positive and negative infinity in order to highly outweigh the respective search when beginning. It also uses them when it detects that beta is less than or equal to the alpha value.

1.3 Evaluation Function

The evaluation function is rather heavy but works by taking the surrounding pieces and and giving them weights, i.e. two pieces together would be multiplied by 10, three piece result by 100, and four piece result by 1000 in order to help ensure it is always selected. It also uses the distances away from similar pieces in a row for example, having two on a line with a single space in between them would result in a combo of 3, which would then be applied the weight. Ultimately it works quite well it seems, although I'm curious as to how it may perform against other evaluation functions.

1.4 Selection / Shortest Win Path

The algorithm always selects the winning move and blocks the opponent's move, which is demonstrated in the following screen shots of game play. In addition, the board is also checked for any winning moves when running the alpha-beta pruning and if one is found for the AI it automatically takes the shortest winning path and delivers it to the algorithm.

2 Results

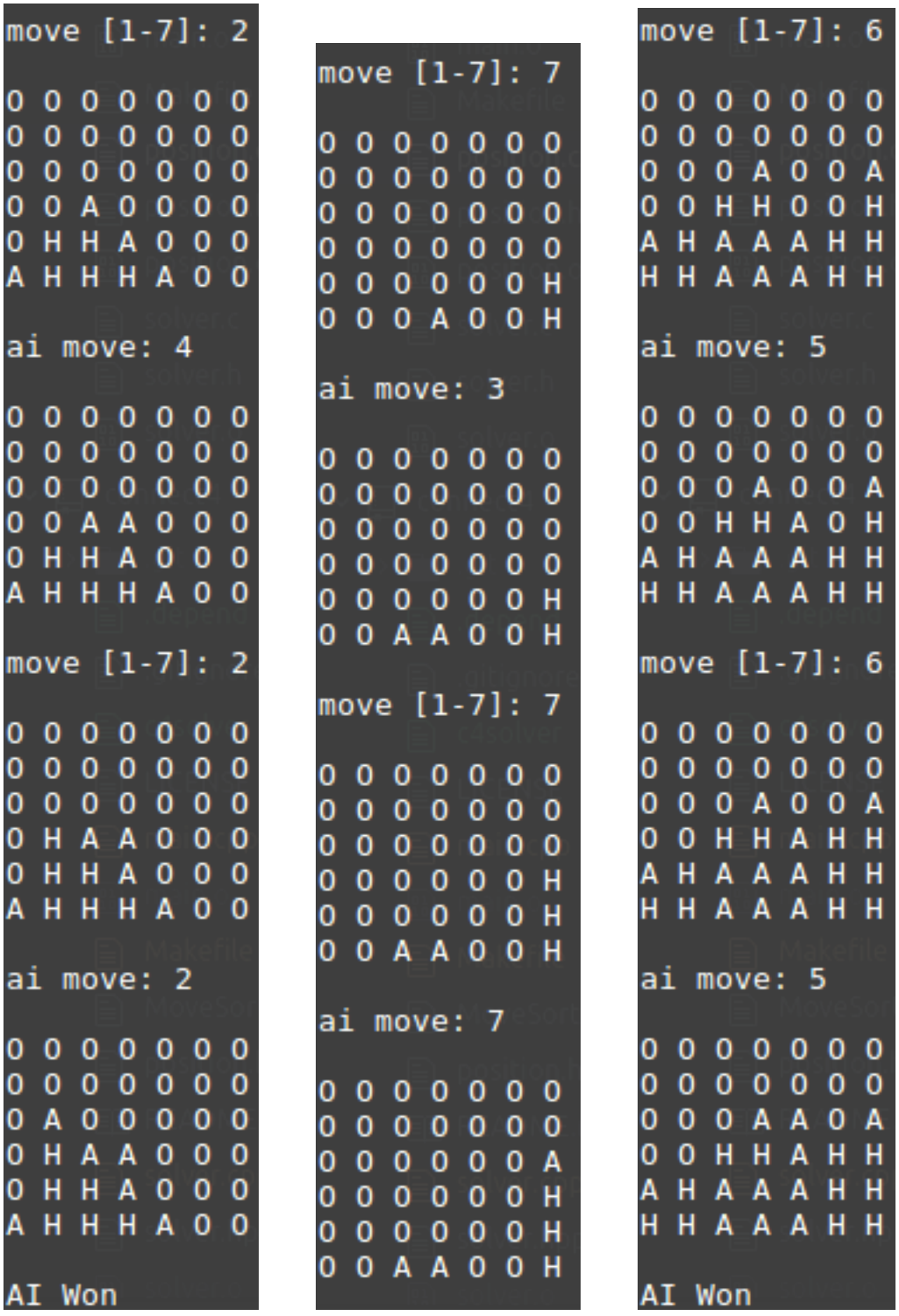

Figure 1: Results from various different game states, described in detail in the next section.

2.1 Game Play

As shown in the above Figure 1, the game state on the left (a) shows the program choosing the winning move in addition to blocking the human from winning at the same time, not bothering to even look at the 3 piece combo on row 4. In the second screenshot (b) the AI blocks the user from winning as soon as it detected a 3 in a row combo. The last screenshot (c) shows the AI favoring a win over blocking the user, because that is a faster method to finding the correct solution. Rather than blocking on row 6, it takes row 5 and wins the game. It's so smart it's almost scary.

3 Conclusion

The program appears to play extremely well when there are diagonal wins, and seems to favor winning diagonally which I found rather interesting. A strength of the program is that it evaluates fairly quickly and is able to pull a reasonable decision from the tree. One weakness is that it is rather weak when playing against a solved connect 4 AI, so in the future I would like to be able to add move ordering and perhaps a table to store previous moves as a hash so it doesn't need to continuously route through the same nodes every time. All in all though I feel pretty happy with how it plays and I am interested to see how it would compare with others.

4 Code Appendix

```
// matt waltz
// connect 4 ai algorithm
// cs470 spring 2018

#include <stdio.h>
#include <stdlib.h>
#include <stdint.h>
#include <stdbool.h>
#include <limits.h>

#define WIDTH 7
#define HEIGHT 6
#define MAX_DEPTH 9
#define max(x, y) (((x) > (y)) ? (x) : (y))
#define min(x, y) (((x) < (y)) ? (x) : (y))

static uint8_t board[HEIGHT][WIDTH];
static int move_next;

enum {
    NONE=-1, EMPTY, AI, HUMAN, CHECK, COMBO=4
};

static char *win_str[] = {
    "Draw Game\n", "AI Won\n", "You Won\n",
};

bool legal(int col) {
    return col < WIDTH && col >= 0 && !board[0][col];
}

void move_col(int col, int player) {
    for (int i=HEIGHT-1; i>=0; i--) {
        if (board[i][col] == EMPTY) {
            board[i][col] = player;
            break;
        }
    }
}

void move_undo(int col) {
    for (int i=0; i<HEIGHT; i++) {
        if (board[i][col]) {
            board[i][col] = EMPTY;
            break;
        }
    }
}

void show_board(void) {
    printf("\n");
    for (int i=0; i<HEIGHT; i++) {
        for (int j=0; j<WIDTH; j++) {
            printf("%c ", board[i][j] == AI ? 'A' : board[i][j] == HUMAN ? 'H' : 'O');
        }
        printf("\n");
```

```
57      }
58      printf("\n");
59  }
60
61  int calc_score(int m, int distance) {
62      int score = COMBO - distance;
63      switch (m) {
64          case 0: return 0;
65          case 1: return 1 * score;
66          case 2: return 10 * score;
67          case 3: return 100 * score;
68          default: return 1000;
69      }
70  }
71
72  int check_board(void) {
73      int k, ai = 0, human = 0;
74      for (int i=HEIGHT-1; i>=0; i--) {
75          for (int j=0; j<WIDTH; j++) {
76              if (j <= CHECK) {
77                  for (k=0; k<COMBO; k++) {
78                          if (board[i][j+k] == AI) ai++;
79                          else if (board[i][j+k] == HUMAN) human++;
80                          else break;
81                  }
82                  if (ai == COMBO || human == COMBO) {
83                      return ai == COMBO ? AI : HUMAN;
84                  }
85                  ai = human = 0;
86              }
87              if (i >= CHECK) {
88                  for (k=0; k<COMBO; k++) {
89                          if (board[i-k][j] == AI) ai++;
90                          else if (board[i-k][j] == HUMAN) human++;
91                          else break;
92                  }
93                  if (ai == COMBO || human == COMBO) {
94                      return ai == COMBO ? AI : HUMAN;
95                  }
96                  ai = human = 0;
97              }
98              if (j <= CHECK && i >= CHECK) {
99                  for (k=0; k<COMBO; k++) {
100                     if (board[i-k][j+k] == AI) ai++;
101                     else if (board[i-k][j+k] == HUMAN) human++;
102                     else break;
103                 }
104                 if (ai == COMBO || human == COMBO) {
105                     return ai == COMBO ? AI : HUMAN;
106                 }
107                 ai = human = 0;
108             }
109             if (j >= CHECK && i >= CHECK) {
110                 for (k=0; k<COMBO; k++) {
111                     if (board[i-k][j-k] == AI) ai++;
112                     else if (board[i-k][j-k] == HUMAN) human++;
113                     else break;
114                 }
```

```
115                 if (ai == COMBO || human == COMBO) {
116                     return ai == COMBO ? AI : HUMAN;
117                 }
118                 ai = human = 0;
119             }
120         }
121     }
122
123     for (int j=0; j<WIDTH; j++) {
124         if (!board[0][j]) return NONE;
125     }
126
127     return EMPTY;
128 }
129
130 int hueristic(void) {
131     int ai = 1, score = 0, empty = 0;
132     int i, c, r, j, k = 0, distance = 0;
133     for (i=HEIGHT-1; i>=0; i--) {
134         for (j=0; j<WIDTH; j++) {
135             if (board[i][j] == EMPTY || board[i][j] == HUMAN) continue;
136             if (j <= CHECK) {
137                 for (k=0; k<COMBO; k++) {
138                     if (board[i][j+k] == AI) ai++;
139                     else if (board[i][j+k] == HUMAN) {
140                         ai = empty = 0; break;
141                     } else empty++;
142                 }
143                 distance = 0;
144                 if (empty) {
145                     for (c=0; c<COMBO; c++) {
146                         for (r=i; r<HEIGHT; r++) {
147                             if (board[r][j+c] == EMPTY) distance++;
148                             else break;
149                         }
150                     }
151                 }
152                 if (distance) {
153                     score += calc_score(ai, distance);
154                 }
155                 ai = 1;
156                 empty = 0;
157             }
158             if (i >= CHECK) {
159                 for (k=0; k<COMBO; k++) {
160                     if (board[i-k][j] == AI) ai++;
161                     else if (board[i-k][j] == HUMAN) {
162                         ai = 0; break;
163                     }
164                 }
165                 distance = 0;
166                 if (ai) {
167                     for (r=i-k+1; r<=i-1; r++) {
168                         if (board[r][j] == EMPTY) distance++;
169                         else break;
170                     }
171                 }
172                 if (distance) {
```

```
173                     score += calc_score(ai, distance);
174                 }
175                 ai = 1;
176                 empty = 0;
177             }
178             if (j >= CHECK) {
179                 for (k=0; k<COMBO; k++) {
180                     if (board[i][j-k] == AI)ai++;
181                     else if (board[i][j-k] == HUMAN) {
182                         ai = empty = 0; break;
183                     } else empty++;
184                 }
185                 distance = 0;
186                 if (empty)
187                     for (c=0; c<COMBO; c++) {
188                         for (r=i; r<HEIGHT; r++) {
189                             if (board[r][j-c] == EMPTY) distance++;
190                             else break;
191                         }
192                     }
193 
194                 if (distance) {
195                     score += calc_score(ai, distance);
196                 }
197                 ai = 1;
198                 empty = 0;
199             }
200             if (j <= CHECK && i >= CHECK) {
201                 for (k=0; k<COMBO; k++) {
202                     if (board[i-k][j+k] == AI) ai++;
203                     else if (board[i-k][j+k] == HUMAN) {
204                         ai = empty = 0; break;
205                     } else empty++;
206                 }
207                 distance = 0;
208                 if (empty) {
209                     for (c=0; c<COMBO; c++) {
210                         for (r=i-c; r<HEIGHT; r++) {
211                             if (board[r][j+c] == EMPTY) distance++;
212                             if (board[r][j+c] == HUMAN) break;
213                         }
214                     }
215                     if (distance) {
216                         score += calc_score(ai, distance);
217                     }
218                     ai = 1;
219                     empty = 0;
220                 }
221             }
222 
223             if (i >= CHECK && j >= CHECK) {
224                 for (k=0; k<COMBO; k++) {
225                     if (board[i-k][j-k] == AI) ai++;
226                     else if (board[i-k][j-k] == HUMAN) {
227                         ai = empty = 0; break;
228                     } else empty++;
229                 }
230                 distance = 0;
```

```
231                 if (empty) {
232                     for (c=0; c<COMBO; c++) {
233                         for (r=i-c; r<HEIGHT; r++) {
234                             if (board[r][j-c] == EMPTY) distance++;
235                             if (board[r][j-c] == HUMAN) break;
236                         }
237                     }
238                     if (distance) {
239                         score += calc_score(ai, distance);
240                     }
241                     ai = 1;
242                     empty = 0;
243                 }
244             }
245         }
246     }
247     return score;
248 }
249
250 int minmax(int depth, int player, int alpha, int beta) {
251     int max = INT_MIN;
252     int min = INT_MAX;
253
254     if (beta <= alpha) {
255         return player == AI ? INT_MAX : INT_MIN;
256     }
257
258     switch (check_board()) {
259         case AI: return INT_MAX - 1;
260         case HUMAN: return INT_MIN + 1;
261         case EMPTY: return 0;
262         default: break;
263     }
264
265     if (depth == MAX_DEPTH) {
266         return hueristic();
267     }
268
269     for (int j=0; j<WIDTH; j++) {
270         if (!legal(j)) continue;
271         int score = 0;
272         switch (player) {
273             default:
274             case AI:
275                 move_col(j, player);
276                 score = minmax(depth + 1, HUMAN, alpha, beta);
277                 if (!depth) {
278                     if (score > max) move_next = j;
279                     if (score == INT_MAX - 1) {
280                         move_undo(j);
281                         return score;
282                     }
283                 }
284                 max = max(score, max);
285                 alpha = max(score, alpha);
286                 break;
287             case HUMAN:
288                 move_col(j, player);
```

```
                score = minmax(depth + 1, AI, alpha, beta);
                min = min(score, min);
                beta = min(score, beta);
                break;
        }
        move_undo(j);
        if (score == INT_MAX || score == INT_MIN) break;
    }
    return player == AI ? max: min;
}

int move_ai(void) {
    minmax(0, AI, INT_MIN, INT_MAX);
    return move_next;
}

void move_human(void) {
    int move, res;
    printf("move [1-7]: ");
    res = scanf("%d", &move);
    while (res != 1 || !legal(move - 1)) {
        printf("invalid.\n\nmove [1-7]: ");
        res = scanf("%d", &move);
    }
    move_col(move - 1, HUMAN);
}

int main(void) {
    int res;
    char input[20];
    printf("play first? [y/n]: ");
    fgets(input, 20, stdin);

    if (input[0] == 'y') {
        move_human();
    }

    show_board();
    int ai_move = move_ai();
    move_col(3, AI);
    show_board();

    for (;;) {
        move_human();
        show_board();
        if ((res = check_board()) != NONE) {
            printf("%s", win_str[res]);
            break;
        }

        int ai_move = move_ai();
        printf("ai move: %d\n", ai_move + 1);

        move_col(ai_move, AI);
        show_board();
        if ((res = check_board()) != NONE) {
            printf("%s", win_str[res]);
            break;
```

```
347             }
348         }
349 
350         return 0;
351 }
```