ITCS 3152-091 Group Project - Connect 4

Jason Pollman, Walter Alfaro, Cullin Moran, Robert Iannucci, Ricky Sanders

Program Instructions

Load “connect4.lisp”, located in the root of the project directory.
Ex: (load “path/to/project/connect4.lisp”)

All other needed files will be loaded automatically and the program will begin execution immediately.

Abstract

Our project’s aim was to create an interactive, AI-driven simulation of the classic two-player game “Connect 4”. The user of the application would play a round of Connect 4 against an artificial player driven by our program. All input and output would happen through a standard system terminal. Our language of choice was Lisp, due to its relative suitability for AI-based applications. The project was divided up into sections, each section being assigned to 1 or 2 members of our group. Programming collaboration was handled by using a Git repository hosted on GitHub.com.

Introduction

AI has been an integral part of computer games since their beginning. A well-crafted, intelligent AI program can make for a challenging—or even impossible to beat—competitor.

Our project will focus on this traditional use of AI by simulating the general game play of the classic board game of Connect Four. Our implementation will be designed for one human player versus a computer player. The computer player will have 3 settings to play against: Easy, Medium, and Hard. Computer AI comes into play to a small amount in the Medium difficulty, but is mostly seen in the Hard difficulty. The computer at easy level will make relatively random, but legal moves. The medium level will attempt to get 4 in a row in the fastest or easiest manner such that it will drop 4 game pieces right next to each other unless blocked by a player. The hard level will block the player’s attempts at getting 4 in a row.

Goals:

- Implement a working version of Connect Four complete with ASCII game board.
- Properly implement Easy, Medium, and Hard AI components to compete with the User.

AI Player Approach

The AI part of our version of Connect 4 worked to challenge the player by getting 4 in a row as quickly as possible and by blocking a player's attempt at getting 4 in a row.

In the easy setting, moves are made *randomly* between Column 1 Row 1 (Move #0) to Column 7 Row 6 (Move #41). However, a move will only be made legally, meaning a move will not be made in a row until it has a piece directly beneath it.

In the intermediate setting, the AI attempts to get four in a row by assessing the moves it's made so far and determining in each direction (horizontally, vertically, or diagonally) if it currently has any laid pieces it can place a new piece beside. This is done by saving the last move made by the AI, then attempting to place a piece next to it in any direction where a move is available. When a piece can be placed next to a previously played piece we called it a "concurrent move." The available concurrent moves are added to a list, and the AI chooses the "concurrent move" closest to the last move it made. The length of the streak of AI moves doesn't matter, only that we can place an AI piece next to another AI piece—that is, since the AI always tries to make a move next to the last piece it made. If there's no way to make a "concurrent move" or it's the first game of the round, the easy function is called and the AI makes a random move.

In the hard setting, the list of game moves is examined by the AI, and the AI determines if the player has 3 in a row in any direction (horizontal, vertical, or diagonal). If the player does have 3 in a row, it attempts to block the player's next potential winning move by determining whether or not that move is available (i.e. the move hasn't already been made and that there are pieces beneath it to support the move). If there's no blocks needed to prevent the player from making the fourth move, the intermediate function is called and the AI attempts to place a piece next to one of its currently played pieces.

Two things that could have been done to enhance the AI in this game are:

1. In the intermediate function, checking to determine if a player has blocked an AI "3-in-a-row" streak. Since the AI always attempts to place a piece next to its last made move, if a player blocks the AI, then it's next move usually ends up right next to the player's block.
2. Since the AI determines its next move by finding the closest move to its last, it always attempts to win horizontally before any other direction. This should, by nature, be random… like a human who might decide to win vertically first. It acts in this manner since the AI loops through the possible "concurrent moves" and picks the one with the least absolute value from the last move (and obviously the move to the left or right will have the lowest absolute value).
3. In the hard setting, the AI will block the user even if it's the AI's move, rather than making a win.

Implementation

The structure of our program revolved around manipulation of a global variable called “game”. This variable was a list which stored moves by the player and the AI. New moves were appended to the list, which was structured *((col, row, player) (col, row, player) ...)*. Columns were numbered 1 - 7, rows were numbered 1 - 6, and for “player”, we used 1 to represent the human player and 0 to represent the AI. Therefore, a sample game board could be imagined visually as follows:

```
6 |_|_|_|_|_|_|_|
5 |_|_|_|_|_|_|_|
4 |_|_|_|_|_|_|_|
3 |_|_|_|_|_|o|_|
2 |_|o|x|_|x|x|_|
1 |o|x|o|o|o|x|_|
   1 2 3 4 5 6 7
```

(where “x” represents the player’s moves and “o” represents the AI’s.)

Work was separated into 5 distinct tasks and was delegated as follows:

- Program Wrapper (connect4.lisp): Jason
- Main method (main.lisp): Jason, Robert, Ricky
- AI functions (ai.lisp): Jason
- Player functions (main.lisp): Robert, Ricky
- Output functions (output.ai): Cullin
- Win-checking functions (winCheck.ai): Walter
- Helper functions (helper.ai): Everyone

The program took this general structure:

1. Call connect4.lisp
2. Load all dependent files (“inc” directory)
3. Files compiled (compile results saved in “tmp/compile.log”)
4. (begin main loop)
    a. Display welcome message
    b. Decide if player/AI plays first (random)
    c. Print (blank) board
    d. Check for a win
    e. Decide move (input from user or AI)
    f. Output board.
    g. If applicable, print win message.
    h. If applicable, prompt user to play again.
5. (end main loop)

Testing Results

Testing of the program revealed some oversights pertaining to certain patterns when checking for a game win. Sometimes the win-check function would prematurely determine that a player had “won” the game when the game should not have ended yet.

Our group also realized that running the program twice during the same Allegro session would result in the game ending immediately on the second try, since the global “game” variable had not been reset. For this reason, we created a “launcher” program (the “connect4.lisp” file) which fixes the issue by clearing out data from previous games.

Another issue we ran into was that the program never seemed to accept input correctly, even though we double and triple-checked our code for errors. It turned out to be a problem with the Allegro environment; updating Allegro resolved the issue.

Summary

Our project was successful; we achieved our goal of creating an AI-based opponent in a Connect 4 game. Git worked well as a source code management system and helped us collaborate better than any other system probably could have.

All parts of the project were completed by the deadline, though we did work up until the very last possible day. Most planning and problem-solving was done during our in-person meetings; the project would have been a lot more difficult to organize had we not met in person to work on it.

Our project’s Git repository/source code can be found at:
https://github.com/JasonAtTheory/3152project

Future Work

1. Finish our group presentation.
2. Implement the changes mentioned above into ai.lisp and make the AI more “natural.”