CSCI 3202
Course Project
October 17, 2025
Points: 210 plus 50 possible extra credit points

Introduction
The course project is designed to let you show what you have learned about artificial intelligence. You have 2 options: you can complete the project as outlined here, or you can propose an alternative project that you will complete and turn in using the same schedule. Once you have chosen an option, you will work alone or in a team with another person (up to two total) to create a project, test it and write up the results.

This project represents 20% of your course grade

Option 1: Your Choice
Create a proposal and submit it by Wed, 10/22. In the proposal, describe 1) what you will do, 2) how it relates to the work you have done in this class (for example, solve a challenging research question using a Bayes Network, create an AI agent that solves a 9-tile game, create an AI player that plays the game of Hex against a computer, or maybe something total different) and 3) how you will test it to determine whether your application works or not.

All deadlines and points totals are the same as for Option 2.

By Wednesday, 11/5, turn in a progress report. This includes 1) your code, 2) the results of your testing so far and an explanation of where you found the libraries and frameworks you used in creating the code. If you code does not work, explain why you believe it doesn't work and how your plan to fix it.

Finally, by Monday, 11/17, create and turn in your final report. In this report, turn in 1) your code, 2) an explanation of what your code does, the libraries and frameworks you used, 3) your test results and a description of how you tested it, how you tested the code and your test results, 4) an example of how your application works and 5) a 2-3 paragraph conclusion about what you learned.

You can earn up to 20 points of extra credit for an explanation of how the Artificial Intelligence works in your project. Write 1-2 pages explaining how and what the Artificial Intelligence does and how you implemented it. You must cite all of your sources in your writeup.

You will be required to participate in interview grading. You will be graded on how well you can explain your project and results from testing.

Option 2: Our Project
Create an AI application that plays Mancala using our modified rules. Test the application as described in this document and create a writeup describing how your application works and shows an example of it working.

Our Mancala Rules
There are many different versions of the rules. This video provides an overview of the rules we are using for this project:

https://youtu.be/OX7rj93m6o8

Our rules may be different from others. We are using the "classic" rules with one exception: if your drop your last stone in your own mancala your turn ends. Do not take another turn. Please read and follow these rules carefully. You may want to watch one or more YouTube videos of Mancala play to help you understand the flow of the game. **You must follow the rules in the video above and described below for the project. **

- Players sit on opposite sides of the long edge of the board
- There are 6 small pits in the middle of the board and 2 large ones at each end. The small ones in the middle and the large pit on your right are yours. The small ones on the other side and the large pit to your opponent's right are theirs
- The large pits at the end of the board are called Mancalas
- Set up the board with 4 stones per small pit (none in the mancalas)
- On every turn, select a pit on your side of the board which contains one or more stones, then distribute its stones, one stone per pit, in an counter-clockwise direction until you have no stones remaining
- If you encounter your opponent's mancala, skip it
- If you encounter your mancala, drop a stone into it
- Do not implement this rule ~~If the last stone you drop is in your own mandala, take another turn immediately~~
- If the last stone lands in an empty pit on your side of the board, capture this stone and any stones in your opponent's pit on the other side of the board, collect all of these stones, including the one that just landed, and place them into your mancala.
- If either player's pits are entirely empty, the game concludes.
- The player who still has stones on his side of the board when the game concludes places all of these pieces into their mancala.
  The player with the most stones in their mancala is declared the winner. If both players have an equal number of stones in their mancala, the game results in a tie.

Project Plan

In HW 6, you will create a Mancala game that allows 2 participants to play our version of Mancala against each other and a single player to play a game of Mancala against a random opponent. We have supplied a framework for this game and provided information on how to construct a random opponent. The instructions and code for HW 6 should help you get started on the project.

We will use this game and random opponent to test your AI code once you have constructed it. This opponent chooses randomly from among the legal moves. With 2 random opponents playing against each other, we would expect each player to win roughly 50% of the time. Once you complete your AI player, it should win more than 50% of the time against a random opponent.

Once you have completed HW 6, you will use the algorithms from the AIMA notebooks and library to construct an AI player for both minimax and alpha beta. Use your notebook from HW 6 to continue this development

Note: If the algorithm you select does not return the best move, you will need to modify it and explain what you modified and why

There are two key ideas we will discuss next week in class, how do we construct a game "tree" and what is an appropriate utility function.

Utility Function

A reasonable utility function is calculated as the number of stones in your Mancala (assuming you are Max) - the number of stones in your opponent's Mancla

Utility=# stones in Max Mancala - # stones in Min Mancala

You are free to use another utility function if you prefer. Remember that you only need to calculate the utility function for terminal nodes. Terminal nodes are those where no further moves are possible (game is won, lost or drawn) or where the tree depth has reached its limit in plies

Game Tree

You will need to evaluate the game tree using minimax or alpha beta in order to choose the next, best move. Minimax and alpha beta are backtracking algorithms, meaning that they start at the top of a tree, descend to the bottom (terminal nodes) then work their way back to the top. The AIMA methods create the game tree as they descend, storing the nodes on the recursion stack. If you use these methods, make sure that you understand how the recursion works and how and where the game tree is stored.

In general, for any state (node) in the tree, there are at most n branches where n is the number of pits on your side of the board (excluding Mancalas). If you are playing on a board with six pits on your side like the one shown in our illustration, there are a maximum of 6 branches. Pits that are empty (have no stones) are not valid moves, limiting the number of branches. If you have 6 pits on your side, but only 2 have stones in them, then there are only 2 valid moves for this state.

You could explicitly construct this tree using the methods from HW 1, or you could construct it in pieces for each level. The algorithms given in the text provide guidance on how to construct and move through this tree.

Project outline

1. (HW 6) Implement an interface that allows you to play Mancala
    - Prints the current state of the game
    - Prompts player to enter a move
    - Determines whether a move is legal or not
    - Determines if someone has won and ends the game
2. (HW 6) Build a random player--a player that makes random (legal) move
3. Play 100 games of random player against random player
    - What percentage of games does each player (1st or 2nd) win?
    - On average, how many moves does it take to win?
    - You should see a small first player advantage in your random against random games
    - We expect the first player to win about 50-55% of the time over 100 games. Since this process is random, the percentage of wins can vary
4. Build an AI player that uses minimax to choose the best move with a variable number of plies and a utility function we describe
    - What percentage of games does each player (AI or random) win?

    - On average, how many moves does it take to win?
5. Play 100 games with the random player against the minimax AI player at a depth of 5 plies
    - What percentage of games does each player (AI or random) win?
    - On average, how many moves does it take to win?
    - Is your AI player better than random chance? Write a paragraph or two describing or why not
6. Build an AI player that uses Alpha-Beta to choose the best move
7. Play 100 games with the random player against the Alpha-Beta AI player at a depth of 5 plies
    - How long does it take for a single game to run to completion?
    - What percentage of games does each player (AI or random) win?
    - On average, how many moves does it take to win?
    - Are your results for this part different from those for your minimax AI player? Write a paragraph or two describing why or why not
8. Play 100 games with the random player against the Alpha-Beta AI player at a depth of 10 plies
    - How long does it take for a single game to run to completion?
    - What percentage of games does each player (AI or random) win?
    - On average, how many moves does it take to win?
    - How much does the Alpha Beta algorithm speed up the game. Compare your run time for 5 ply minimax against 5 ply Alpha Beta. Project how long Minimax would take to run 10 plies.
    - Plot a curve showing the win percentage for a player looking ahead 2 plies, 5 plies and 10 plies
    - As you increase the number of plies, does the AI player win more games? Explain why or why not.
9. (Extra Credit, 15 points). Change the utility function and play 100 games with the random player against the Alpha-Beta AI player at a depth of 10 plies or more
    - How long does it take for a single game to run to completion?
    - What percentage of games does each player (AI or random) win?
    - On average, how many moves does it take to win?
    - In your writeup, explain how your new utility function improves on the utility function described above?
    - Explain how increasing the number of plies improve the play for the AI player?
    - Is this new utility function a better way to evaluate the strength of a

particular match? Explain how?

10. (Extra Credit, 35 points). Implement the continuation rule fully. If your drop your last stone in your own mancala you take another turn. Continue taking additional turns as long as your last stone drops into your mancala.
    - Show the code changes necessary to make this rule work
    - Compare the game results for 10 plies with Alpha Beta with and without the continuation rule
    - Explain what causes the win rate to be different.

Key Dates and Points Awarded

- **Fri, October 17, Project Assigned**
  - Project Assigned
- **Wed, October 22, Project Proposal and Team**
  - Alternate Project Proposal Submitted (If Desired) or Declare You are Using the Default Project (10 points)
  - Partner Name Submitted (At most 2 people)
- **Wed, 11/5, Intermediate Results**
  - Intermediate Results Due (50 points)
  - For full credit, I expect that you will have completed all of the tasks in HW #6, and have implemented and tested random vs. random players for 100 games
  - Turn in code, a list of the libraries and frameworks you used to create the code and the results of your testing so far
  - If your code does not run, be prepared to explain why you believe it doesn't run and how you are planning to fix it. **You still must turn in a notebook with your code**
  - Turn in your code and writeup in an ipynb file
- **Mon, 11/17, Project Due**
  - Turn in your project code and writeup (100 points)
  - Turn in an ipynb file with your code. Your code should run as described in your writeup.
- **Wed, 11/19-11/21 or Mon, 12/1-12/5, Interview Grading**
  - 15 minute interview (50 points)
  - Have your code running on a computer that can share its screen over Zoom
  - Be prepared to run your code with a variety of configurations, show your code, and describe how the components of your code works