Faculty of Engineering-Ain Shams University
Computer and Systems Engineering Department

CSE481: Artificial Intelligence

Mancala Game

Team Members:

Name	Code
احمد عبد العظيم حمدى سعداوى	1600118
مروان محمود بدوى	1603416
شريف محمد عبدالرحمن عبدالجواد	1600694
محمد محمود عبدالله احمد الانصارى	1601269

Description

Mancala is a two player turn based strategy board game played with small stones. There are 6 pits at each player's side and one store for each player. The objective of the game is to collect as many stones in your store as possible. The player with the most stones in his/her store at the end of the game wins.
Game support different difficulty levels based on the tree depth. Game support different mode Stealing mode and Non-Stealing mode

Rules

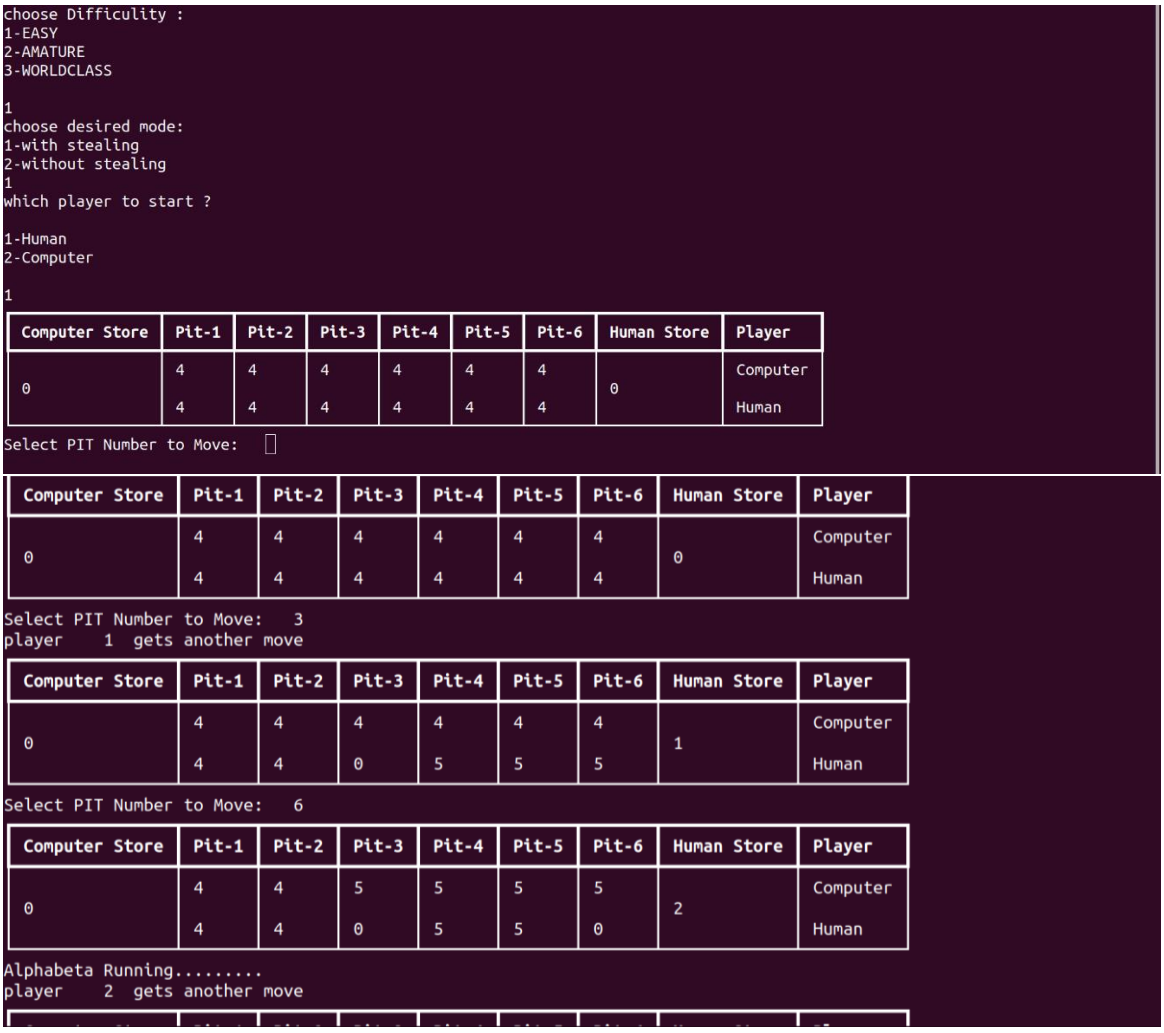
- Initially there are 4 stones in each pit and stores are empty.
- Play always moves in a counterclockwise circle (left to right).
- Each player controls 6 pits at his/her side.
- When a player makes a move, the player takes all stones from one of the pits under the player's control. Moving counterclockwise the player drops one stone in each pit in a turn, including the player's own store but not their opponent.
- If the last stone landed in your store, take another move.
- If the last stone landed in one of your pits and this pit was empty one, you take all stones in your opponent's pit opposite to yours.
- The game is over when one player's pits are empty. The other player takes the seed from his pits and puts them in his store and counts up the stones. Whoever has the most stones win.

Tasks

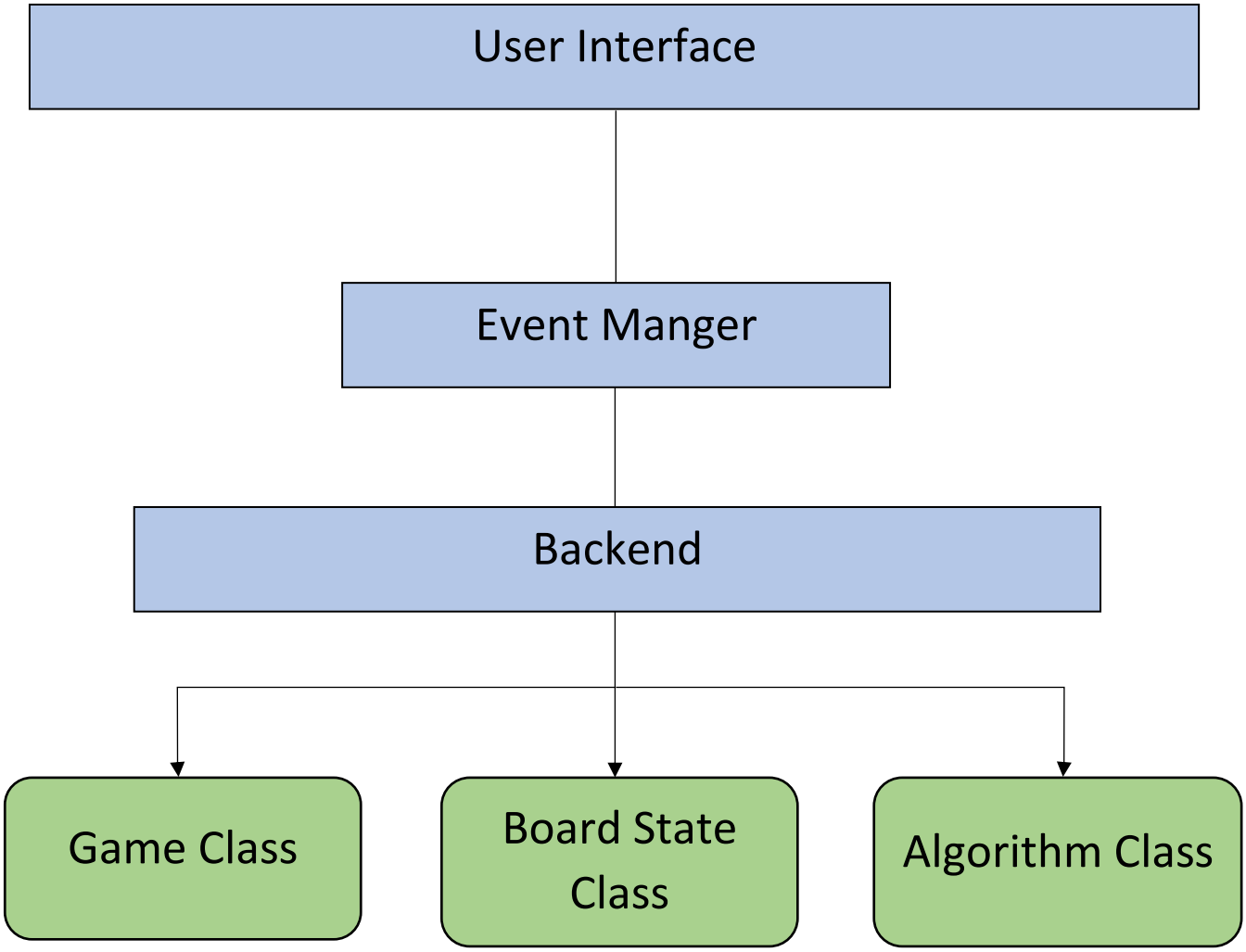
Task	Members
Board class	محمد محمود عبدالله احمد الانصارى مروان محمود بدوى
Algorithm class	احمد عبد العظيم حمدى شريف محمد عبد الرحمن عبد الجواد
Integration & play class	احمد عبد العظيم حمدى سعداوى مروان محمود بدوى شريف محمد عبدالرحمن عبدالجواد محمد محمود عبدالله احمد الانصارى
Documentation	احمد عبد العظيم حمدى سعداوى مروان محمود بدوى شريف محمد عبدالرحمن عبدالجواد محمد محمود عبدالله احمد الانصارى
Deployment	محمد محمود عبدالله احمد الانصارى

User Guide

1. Choose Difficulty level by enter (1, 2, 3)
    1. EASY
    2. AMATURE
    3. WORLDCLASS
2. Choose the Mode you desired to play with by entering (1,2)
    1. With Stealing
    2. Without stealing
3. Choose which player to start play by entering (1,2)
    1. Human
    2. Computer
4. When your turn is come then enter the number of PIT you want to move



System Architecture



User Interface:

- Command line interface showing the table of each state
- Taking the next move from the player

Event Manger:

- Whenever player enter his move, passing the required information to the backend

Backend:

- Consist of 3 Main classes (Game, Board_State, Algorithm)
- Contains the actual Implementation

Functions Description

1. Game Class

Function Name	Input	Output	Description
play	**playerStrategy**: defines the human and algorithm strategy **player**: player whose turn is now **board**: created object of BOARD class **difficulty**: chosen difficulty of the game by the user Easy ---> difficulty =1, AMATURE ---> difficulty =2 WORLDCLASS --> difficulty =3 **mode:** chosen mode by user 1--> stealing(mode=true) 2--->without stealing(mode=false)	bool	1. set depth according to difficulty 2. make moves of stones on board either by human or algorithm 3. Check if game is terminated
start	**playerStrategy1**: defines the human or algorithm strategy **playerStrategy2**: defines the human or algorithm strategy **difficulty**: chosen difficulty of the game by the user Easy ---> difficulty =1, AMATURE ---> difficulty =2, WORLDCLASS --> difficulty =3 **mode:** chosen mode by user 1--> stealing(mode=true) 2--->without stealing(mode=false)	.....	Game loop

2. Board State Class

Function Name	Input	Output	Description
getLimit	**player**: current player	integer	Calculating the index of starting PIT according to current player
evaluation	**player**: current player	….	Evaluate the current state based on 1. Difference between the two stores 2. the Difference between number of stones in each PIT and it's opposite PIT
Move	**player**: current player **move:** index of chosen PIT to move stones from it **mode:** stealing (1) or without stealing(0)	bool	Actual move of stones from chosen PIT to adjacent PIT
getScore	**player**: current player	integer	Calculate score from both player's stores to reveal the game result (absolute difference between stores)
isGameOver		bool	Checks if the game is over when one player's pits are completely empty.
print_board		….	Responsible for printing the game board

3. Algorithm Class

Function Name	Input	Output	Description
calcMaxValue	**board_state:** object of BOARD class **player**: current player **depth:** chosen depth **alpha** **beta** **mode:** chosen mode (stealing or without stealing)	integer	Calculate Max Value of the current node
calcMinValue	**board_state:** object of BOARD class **player**: current player **depth:** chosen depth **alpha** **beta** **mode:** chosen mode (stealing or without stealing)	integer	Calculate Max Value of the current node
alphabetaAlgorithm	**player**: current player **depth:** chosen depth **mode:** chosen mode (stealing or without stealing) **board_state:** object of BOARD class	integer	Apply AlphaBeta algorithm using recursion And return index of PIT to be moved

Description of utility function:

Evaluate is based on

1. Difference between the two stores
2. Difference between number of stones in each PIT of player1 and number of stones in in each PIT of player2