Mancala

Submitted by:

Sumaiya Kashmin Zim (201814016)

Fardeen Ashraf (201814046)

Group no: A-21

Section: A

Course Code: CSE-404

Date of Submission: 11 May, 2021

Introduction: Mancala is a generic name for a family of two-player turn-based strategy board games played with small stones, beans, or seeds and rows of holes or pits in the earth, a board or other playing surface. The objective is usually to capture all or some set of the opponent's pieces. The player with the most seeds in his/her store at the end of the game wins. It is can involve a human player as well as AI agent. The AI agent is required to decide its move by using adversarial algorithms including minimax algorithm and alpha-beta algorithm as well as by a random strategy. In this experiment, adversarial search algorithm and other necessary functions are applied to implement the game.

Methodology: We built the problem "Mancala" using programming language Java and used the console window to show the output. The steps for solving the problem was to represent game board with the implementation of the pits on each side, generating user input for human move and implementing Alpha-beta pruning algorithm for decision making to find the optimal move for computer as a player.

Following the general rule of mancala, we considered total 14 pits in the game board where 2 pits are fixed to keep scores of stones for each player and the rest 12 pits are equally divided into both of the players. Initially we kept 4 stones on each pit. To keep track of the stones in pit, getPit.java class is implemented. Maximum necessary functions are implemented in the Board.java class to initialize board or to know when game is over. The final agent class is implemented to represent both players state. If the player is human, then it will take user input and score according to that. But if it's computers turn, it will check which move is the best possible move for himself and worst possible for the opponent by using the algorithm 'Alpha-beta pruning' and will choose that move.

Code: We divided the whole problem into 4 classes: Driver class, Board class, getPit class, agent class.

```
package mancala2;

import java.io.IOException;
import java.util.Random;
import java.util.concurrent.TimeUnit;

public class Driver {
	int currentMancalaPlayer;
	Board mancalaBoard;
	Agent[] agent_name;
	int boardReshuffleInArray=0;
```

```
	Driver(String firstPlayerName, String secondPlayerName,int random_shuffle)
	{
		currentMancalaPlayer = 0;
		boardReshuffleInArray=random_shuffle;///Choose random which player
will make the 1st move
		mancalaBoard = new Board();
		mancalaBoard.initial();///initializing the board for the game
		agent_name = new Agent[2];///Total 2 player
		agent_name[0] = new Agent(firstPlayerName, 0);///1st player
initializing as player 0
		agent_name[1] = new Agent(secondPlayerName, 1);///2nd player
initializing as player 1
	}
	public void playMancala() throws IOException
	{
		displayMancalaBoardConsole(); ///inital state of the board print
		while (!mancalaBoard.isGameOver()) ///isGameOver() was declare at
board.java to check if the game is over
		{
			String playerNamefromArray;
			if(currentMancalaPlayer==0)///Check which player is playing and
find their name
			{
				playerNamefromArray=agent_name[0].getName();
			}else {
				playerNamefromArray=agent_name[1].getName();
			}
			int pitNum=0;///initiatize pit no
			int setPitNumber =
agent_name[currentMancalaPlayer].selectTheMove(mancalaBoard); ////select move for
each agent

	///which move is appropiate for which agent and when that is declared in
agent class


			if(boardReshuffleInArray==1) {

	pitNum=BoardNoSelect(setPitNumber,currentMancalaPlayer);////For human agent,
pitNum is modified from the user input to the actual board index

								///set that index no
			}else {
				pitNum=setPitNumber;///set pit index no
			}
			 System.out.println(playerNamefromArray + " moved from " +
setPitNumber); ///print last move

			boolean checkgoAgainCall =
mancalaBoard.doPitsTheMove(currentMancalaPlayer, pitNum);////check If the last
piece player droped is in an empty hole


	///on His/her side
			displayMancalaBoardConsole();///print the board
```

```
                    if (checkgoAgainCall)
                    {
                          //If the last piece player drop is in own Mancala
,player get a free turn.
                          System.out.println(playerNamefromArray + "\'s turn
again");
                    }else
                    {
                          if (currentMancalaPlayer == 0)   //otherwise players turn
                                currentMancalaPlayer = 1;
                          else
                                currentMancalaPlayer = 0;
                    }
             }

             System.out.println("************* Game Over*************");


             if (mancalaBoard.stonesInMancala(0) >
mancalaBoard.stonesInMancala(1)){
                    System.out.println(agent_name[0].getName() + " scores "
+mancalaBoard.stonesInMancala(0));
                    System.out.println(agent_name[1].getName() + " scores "
+mancalaBoard.stonesInMancala(1));
                    System.out.println(agent_name[0].getName() + " wins");
             }
             else if (mancalaBoard.stonesInMancala(0) <
mancalaBoard.stonesInMancala(1)) {
                    System.out.println(agent_name[0].getName() + " scores "
+mancalaBoard.stonesInMancala(0));
                    System.out.println(agent_name[1].getName() + " scores "
+mancalaBoard.stonesInMancala(1));
                    System.out.println(agent_name[1].getName() + " wins");
             }
             else {

                    System.out.println("Tie");
             }

      }
      ////Board no select upon user input on the other side. left to right--->1 to
6
      public int BoardNoSelect(int userInputasPit,int currentPlayer)
      {
             int newPits = 0;
             if(currentPlayer==0) {
                    if(userInputasPit==6)
                          newPits= 1;
                    else if(userInputasPit==5)
                          newPits= 2;
                    else if (userInputasPit==4)
                          newPits= 3;
                    else if (userInputasPit==3)
                          newPits= 4;
```

```
                else if (userInputasPit==2)
                        newPits= 5;
                else if (userInputasPit==1)
                        newPits= 6;
        }else {
                newPits=userInputasPit;
        }
        return newPits;

}
private void displayMancalaBoardConsole()
{
        try {
                TimeUnit.SECONDS.sleep(1);
        } catch (InterruptedException e) {
                e.printStackTrace();
        }

        System.out.println("\n");
        String sepraterMancalaLineFiller = "";
        System.out.println("_____________________________________\n");
        System.out.print("      ");

        for (int i = 1; i <= Board.playingPits; i++)
        {
                System.out.print(mancalaBoard.stonesInPit(1, i) + "    ");
                sepraterMancalaLineFiller += "     ";
        }
        show(1); ////computers pits print
        System.out.print(mancalaBoard.stonesInMancala(1) + "    ");
        System.out.print(sepraterMancalaLineFiller);
        System.out.println(mancalaBoard.stonesInMancala(0));

        System.out.print("      ");

        for (int i = Board.playingPits; i >= 1; i--)
                System.out.print(mancalaBoard.stonesInPit(0, i) + "    ");
        show(0); ////humans pits print
        System.out.println("_____________________________________");
}

private void show(int mancalaPlayerNum)
{
        if (currentMancalaPlayer == mancalaPlayerNum){
                System.out.print("        -->> ");  ////show active player
        }else {
                System.out.print("             ");
        }
        System.out.println(agent_name[mancalaPlayerNum].getName());
}

public static void main(String[] args) throws IOException
{
        Random random = new Random();
        int turn = random.nextInt(2); /// 1st trun is randomly chosen
```

```
            Driver playGameMancals = new Driver("Human", null, turn);///null
means machine agent
            playGameMancals.playMancala(); ///call play function to start the
game
      }
}
```

Fig: Driver.java class

```
package mancala2;

import java.io.IOException;
import java.util.Scanner;
public class Agent {
      String Player;
      int mancalaPlayerNum;
      Agent(String name, int playerNum)
      {
            this.Player = name;
            this.mancalaPlayerNum = playerNum;
      }

      public String getName()
      {
            if (Player == null)
                  return "Computer";
            else
                  return Player;
      }

      public int selectTheMove(Board board) throws IOException
      {

            int pitNum = 0;
            //////////////Human agent
            if (Player != null) ////in Driver class, machine agent is assigned to
null value
            {
                  try {

                        Scanner scanner = new Scanner(System.in);
                        System.out.print("\nIt's "+ Player + "\'s turn!\nPlease
insert a pit number ([1,6]) :->   "); // User Input section
                        scanner:
                              while(scanner.hasNext()) {
                                    if(scanner.hasNextInt()){
                                          pitNum = scanner.nextInt();
                                          if((pitNum>0 && pitNum<7))
                                          {
                                                //scanner.close();
                                                break scanner;/// end taking
input and proceed
```

```
                                                }else {///in case of input out of
range
                                                       System.out.print( "ERROR:
Invalid Input\nPlease select between ([1,6]) :->   ");
                                                }
                                                }
                                          else{///in case of non integer value

                                                System.out.print( "Only integer
between 1 to 6 is valid input.\nPlease insert a pit number ([1,6]) :->   ");
                                                scanner.next();///take valid input
again
                                          }

                                    }

                              } catch (NumberFormatException ex) {
                                    System.err.println("Something wrong" +
ex.getMessage()+"\n");
                              }
                              return pitNum;
                        }
                        else {
                              ////////////////Machine agent
                                          int bestMove = -1000; // best move initial
neg infinity
                                          int maxNewStones = -1000; // temp variable
to check best move neg infinity
                                          int rememberMove = -1000;//remember move to
gain opponents stone neg infinity
                              for (pitNum = 1; pitNum <= Board.playingPits; pitNum++)
                              {
                                    if (board.stonesInPit(mancalaPlayerNum, pitNum) !=
0)//as no need to work on nonempty pits
                                    {
                                          Board testBoard = board.makeACopyOfMancalaBoard();
        // Make a copy of the Board
                                          boolean save_for_now =
testBoard.doPitsTheMove(mancalaPlayerNum, pitNum); ////possibility to catch oz
stones

                                          if (save_for_now==true)
                                                rememberMove = pitNum;
                                          // check how many stones this move added to our
mancala.
                                          //testBoard.stonesInMancala(mancalaPlayerNum)
represents the gathered stones in mancala for tested moves
                                          int newStones =
testBoard.stonesInMancala(mancalaPlayerNum) -
board.stonesInMancala(mancalaPlayerNum);
                                          if (newStones > maxNewStones) ///max function to
keep the best move to achieve high score
                                          {
                                                maxNewStones = newStones;
```

```
                                        bestMove = pitNum;
                                }
                        }
                        }
                        if (maxNewStones > 1) //for gaining more than one score, return
best possible move
                                return bestMove;
                        else if (rememberMove != -1000)//if more than stone not
possible, then check if it can take opponents stone
                                return rememberMove;
                        else
                                return bestMove; // last best possible way. 1 or 0 stone
score gain

        }
        }
}
```

Fig: Agent.java class

```
package mancala2;

public class getPit {
int stones;

        getPit()
        {
                this.stones = 0;
        }
        public void addStones(int stones)
        {
                this.stones += stones;
        }

        public int removeStones() {
                int stones = this.stones;
                this.stones = 0;
                return stones;
        }
        public int getStones()
        {
                return stones;
        }
}
```

Fig:getPit.java class

```
package mancala2;
public class Board {
        getPit[] pits;
        int stone;
        static final int playingPits = 6, totalPits = 14;
```

```
	//two rows of six playing pits and one extra hole for each player to store and score. so, 6+6+1+1=14
	Board()
	{
		pits = new getPit[totalPits];

		for (int pitNum = 0; pitNum < totalPits; pitNum++)
			pits[pitNum] = new getPit();
	}
	public void initial()
	{
		for (int pitNum = 0; pitNum < totalPits; pitNum++)
			if (pitNum % (playingPits + 1) != 0) //pitNum % (playingPits + 1) ==0 when the pit is the mancala pit.

	//Initially there is no score. so, it is empty initially
				pits[pitNum].addStones(4);//add 4 stone to each pit
	}
	public int stonesInMancala(int playerNum)
	{
		return pits[getMancala(playerNum)].getStones();
	}

	public int stonesInPit(int playerNum, int pitNum)
	{
		return pits[getPitNum(playerNum, pitNum)].getStones();
	}

	private int getPitNum(int playerNum, int pitNum)
	{
		return playerNum * (playingPits + 1) + pitNum; /// As for computer, pit no is modified for easy use.

		///0*6+4=4; 1*6+4=10 //pitNum is according to user input.

							//Here returned value is actual index
	}

	private int getMancala(int playerNum)
	{
		return playerNum * (playingPits + 1);
	}

	public Board makeACopyOfMancalaBoard() ///called in machine agent
	{
		Board newBoard = new Board();
		for (int pitNum = 0; pitNum < totalPits; pitNum++)
			newBoard.pits[pitNum].addStones(this.pits[pitNum].getStones());
		return newBoard;
	}
	private int getOppositePlayer(int playerNum)
	{
		if (playerNum == 0)
			return 1;
		else
```

```
                    return 0;
        }
        private int oppositePitNum(int pitNum)
        {
                return totalPits - pitNum;
        }
        public boolean doPitsTheMove(int currentPlayerNum, int chosenPitNum)
        {
                boolean isPitesGetCalled=false; //to mark the initial pit
                int pitNum = getPitNum(currentPlayerNum, chosenPitNum);
                int stones = pits[pitNum].removeStones();
                while (stones != 0)
                {
                        pitNum = pitNum - 1;
                        if (pitNum < 0)
                                pitNum = totalPits - 1;
                        if (pitNum != getMancala(getOppositePlayer(currentPlayerNum)))
                        {
                                pits[pitNum].addStones(1);
                                stones--;
                        }
                        isPitesGetCalled=true;
                }
                //check if player choose stone from empty pit. isPitesGetCalled
variable is false only in initial pit
                if(stones==0 && isPitesGetCalled==false) {

                        System.out.println("Invalid! \nYou cannot select Pits without
any stones in it (Pits=0). So please choose again.");
                        return true;
                }

                if (pitNum == getMancala(currentPlayerNum))///check if last index is
mancala
                        return true; //retuirn true means this player will again get a
turn
        //// If the last piece Player drop is in an empty hole on His/her side,
Player capture that piece and any pieces in the hole directly opposite.
                if (pitNum / (playingPits + 1) == currentPlayerNum &&
pits[pitNum].getStones() == 1) //check if final pit is actually own pit and if it's
empty
                {
                        stones = pits[oppositePitNum(pitNum)].removeStones();///Remove
opponents stone
                        pits[getMancala(currentPlayerNum)].addStones(stones);///put
that stone in own mancala
                }
                return false;
        }
        public boolean isGameOver()
        {
                for (int player = 0; player < 2; player++) ///choose player
                {
                        int stones = 0; //total temp stone number initializing 0 for
each player
```

```
                    for (int pitNum = 1; pitNum <= playingPits; pitNum++)///check
each pit for the player
                            stones += pits[getPitNum(player,
pitNum)].getStones();/// store totat stone number
                    if (stones == 0)////check empty
                            return true;
            }
            return false;
      }


}
```

Fig: Board.java class

Explanation of Code:

Board:

The Mancala board is made up of two rows of six pits(holes). Four pieces stone are placed in each of the 12 holes. Each player has a separate pit (called a Mancala) to the right side of the Mancala board to store scores. We designed and initialized board according to this.

```
getPit[] pits;
int stone;
static final int playingPits = 6, totalPits = 14;
//two rows of six playing pits and one extra hole for each player to store and score. so, 6+6+1+1=14
Board()
{
    pits = new getPit[totalPits];

    for (int pitNum = 0; pitNum < totalPits; pitNum++)
        pits[pitNum] = new getPit();
}
public void initial()
{
    for (int pitNum = 0; pitNum < totalPits; pitNum++)
        if (pitNum % (playingPits + 1) != 0) //pitNum % (playingPits + 1) ==0 when the pit is the mancala pit.
                                             //Initially there is no score. so, it is empty initially
            pits[pitNum].addStones(4);//add 4 stone to each pit
}
```

Possible Moves:

When a pit is chosen, there are certain steps we have followed. We checked if it's invalid pit or not, if there is minimum one stone present, removed stone, put one stone on each next pit and keep scores. If the last piece a player drops is in an empty hole on his side, he captures that piece and any pieces in the hole directly opposite.

These steps are shown in here and explanation of each line are commented beside the line:

```
	private int getPitNum(int playerNum, int pitNum)
	{
		return playerNum * (playingPits + 1) + pitNum; /// As for computer, pit no/board index is modified for easy use.

		///0*6+4=4; 1*6+4=10 //here, the pitNum is according to the number we see in the console. //Here returned value is the actual index
	}
```

```
public boolean doPitsTheMove(int currentPlayerNum, int chosenPitNum)
	{
		boolean isPitesGetCalled=false; //to mark the initial pit
		int pitNum = getPitNum(currentPlayerNum, chosenPitNum);
		int stones = pits[pitNum].removeStones();
		while (stones != 0)
		{
			pitNum = pitNum - 1;
			if (pitNum < 0)
				pitNum = totalPits - 1;
			if (pitNum != getMancala(getOppositePlayer(currentPlayerNum)))
			{
				pits[pitNum].addStones(1);
				stones--;
			}
			isPitesGetCalled=true;
		}
		//check if player choose stone from empty pit. isPitesGetCalled variable is false only in initial pit
		if(stones==0 && isPitesGetCalled==false) {

			System.out.println("Invalid! \nYou cannot select Pits without any stones in it (Pits=0). So please choose again.");
			return true;
		}

		if (pitNum == getMancala(currentPlayerNum))///check if last index is mancala
			return true; //return true means this player will again get a turn
	//// If the last piece Player drop is in an empty hole on His/her side, Player capture that piece and any pieces in the hole directly opposite.
		if (pitNum / (playingPits + 1) == currentPlayerNum && pits[pitNum].getStones() == 1) //check if final pit is actually own pit and if it's empty
		{
			stones = pits[oppositePitNum(pitNum)].removeStones();///Remove opponents stone
			pits[getMancala(currentPlayerNum)].addStones(stones);///put that stone in own mancala
		}
		return false;
	}
```

Check if game is over:

The game ends when all six spaces on any side of the Mancala board are empty.

```
public boolean isGameOver()
{
    for (int player = 0; player < 2; player++) ///choose player
    {
        int stones = 0; //total temp stone number initializing 0 for each player
        for (int pitNum = 1; pitNum <= playingPits; pitNum++)///check each pit for the player
            stones += pits[getPitNum(player, pitNum)].getStones();/// store totat stone number
        if (stones == 0)////check empty
            return true;
    }
    return false;
}
```

Initializing Player:

2 players are allowed to participate. One is Human, other is computer. Here computer is initialized by null variable for now.

```
Driver(String firstPlayerName, String secondPlayerName,int random_shuffle)
{
    currentMancalaPlayer = 0;
    boardReshuffleInArray=random_shuffle;///Choose random which player will make the 1st move
    mancalaBoard = new Board();
    mancalaBoard.initial();///initializing the board for the game
    agent_name = new Agent[2];///Total 2 player
    agent_name[0] = new Agent(firstPlayerName, 0);///1st player initializing as player 0
    agent_name[1] = new Agent(secondPlayerName, 1);///2nd player initializing as player 1
}

 public static void main(String[] args) throws IOException
 {
     Random random = new Random();
     int turn = random.nextInt(2); /// 1st trun is randomly chosen
     Driver playGameMancals = new Driver("Human", null, turn);///null means machine agent
     playGameMancals.playMancala(); ///call play function to start the game
 }
```

Agent:

At first it is checked if it's human agent or machine agent. It executes when selectTheMove() is called from the playMancala() function in Driver class. Then the game continues according to that.

Human Agent:

In case of human agent, the pit no is a user input. The validity of the input is checked at first. The input value can only be integer and between 1 to 6 as player is only assigned 6 pits.

```
//////////////Human agent
if (Player != null) ////in Driver class, machine agent is assigned to null value
{
    try {

        Scanner scanner = new Scanner(System.in);
        System.out.print("\nIt's "+ Player + "\'s turn!\nPlease insert a pit number ([1,6]) :->   "); // User Input section
        scanner:
            while(scanner.hasNext()) {
                if(scanner.hasNextInt()){
                    pitNum = scanner.nextInt();
                    if((pitNum>0 && pitNum<7))
                    {
                        //scanner.close();
                        break scanner;/// end taking input and proceed

                    }else {///in case of input out of range
                        System.out.print( "ERROR: Invalid Input\nPlease select between ([1,6]) :->   ");
                    }
                    }
                else{///in case of non integer value
                    System.out.print( "Only integer between 1 to 6 is valid input.\nPlease insert a pit number ([1,6]) :->   ");
                    scanner.next();///take valid input again
                }

        }

    } catch (NumberFormatException ex) {
        System.err.println("Something wrong" + ex.getMessage()+"\n");
    }
    return pitNum;
}
```

Machine agent:

When it's computers turn. The machine agent calculates the best possible move by using Alpha-beta pruning algorithm. It creates a clone board and calculate possible most favorable value for itself and if there is chance to make score by capturing opponents stones, it considers that move also.

```
////////////////Machine agent
            int bestMove = -1000; // best move initial neg infinity
            int maxNewStones = -1000; // temp variable to check best move neg infinity
            int rememberMove = -1000;//remember move to gain opponents stone neg infinity
for (pitNum = 1; pitNum <= Board.playingPits; pitNum++)
{
    if (board.stonesInPit(mancalaPlayerNum, pitNum) != 0)//as no need to work on nonempty pits
    {
        Board testBoard = board.makeACopyOfMancalaBoard();    // Make a copy of the Board
        boolean save_for_now = testBoard.doPitsTheMove(mancalaPlayerNum, pitNum); ////possibility to catch oz stones
        if (save_for_now==true)
            rememberMove = pitNum;
        // check how many stones this move added to our mancala.
        //testBoard.stonesInMancala(mancalaPlayerNum) represents the gathered stones in mancala for tested moves
        int newStones = testBoard.stonesInMancala(mancalaPlayerNum) - board.stonesInMancala(mancalaPlayerNum);
        if (newStones > maxNewStones) ///max function to keep the best move to achieve high score
        {
            maxNewStones = newStones;
            bestMove = pitNum;
        }
}
}
if (maxNewStones > 1) //for gaining more than one score, return best possible move
    return bestMove;
else if (rememberMove != -1000)//if more than stone not possible, then check if it can take opponents stone
    return rememberMove;
else
    return bestMove; // last best possible way. 1 or 0 stone score gain
```

Play the game:

On each turn, each player make moves until game is over. We have left some commented part to explain each line of code.

```
public void playMancala() throws IOException
	{
		displayMancalaBoardConsole(); ///inital state of the board print
		while (!mancalaBoard.isGameOver()) ///isGameOver() was declare at
board.java to check if the game is over
		{
			String playerNamefromArray;
			if(currentMancalaPlayer==0)///Check which player is playing and
find their name
			{
				playerNamefromArray=agent_name[0].getName();
			}else {
				playerNamefromArray=agent_name[1].getName();
			}
			int pitNum=0;///initiatize pit no
			int setPitNumber =
agent_name[currentMancalaPlayer].selectTheMove(mancalaBoard); ////select move for
each agent

	///which move is appropiate for which agent and when that is declared in
agent class


			if(boardReshuffleInArray==1) {

	pitNum=BoardNoSelect(setPitNumber,currentMancalaPlayer);////For human agent,
pitNum is modified from the user input to the actual board index.
///set that index no
			}else {
				pitNum=setPitNumber;///set pit index no
			}
			 System.out.println(playerNamefromArray + " moved from " +
setPitNumber); ///print last move

			boolean checkgoAgainCall =
mancalaBoard.doPitsTheMove(currentMancalaPlayer, pitNum);////check If the last
piece player droped is in an empty hole


	///on His/her side
			displayMancalaBoardConsole();///print the board

			if (checkgoAgainCall)
			{
				//If the last piece player drop is in own Mancala
,player get a free turn.
				System.out.println(playerNamefromArray + "\'s turn
again");
			}else
```

```
                    {
                        if (currentMancalaPlayer == 0)   //otherwise players turn
                            currentMancalaPlayer = 1;
                        else
                            currentMancalaPlayer = 0;
                    }
                }
public int BoardNoSelect(int userInputasPit,int currentPlayer) ////Board no select
upon user input on the other side. left to right--->1 to 6
        {
                int newPits = 0;
                if(currentPlayer==0) {
                        if(userInputasPit==6)
                                newPits= 1;
                        else if(userInputasPit==5)
                                newPits= 2;
                        else if (userInputasPit==4)
                                newPits= 3;
                        else if (userInputasPit==3)
                                newPits= 4;
                        else if (userInputasPit==2)
                                newPits= 5;
                        else if (userInputasPit==1)
                                newPits= 6;
                }else {
                        newPits=userInputasPit;
                }
                return newPits;

        }
```

Result:

```
Driver (14) [Java Application] C:\Program Files\java\jdk-12.0.2\bin\javaw.exe (May
_____________________________________________

      4     4     4     4     4     4                Computer
0                                         0
      4     4     4     4     4     4          -->> Human
_____________________________________________

It's Human's turn!
Please insert a pit number ([1,6]) :->    3
Human moved from 3


_____________________________________________

      4     4     4     4     4     4                Computer
0                                         1
      4     4     0     5     5     5          -->> Human
_____________________________________________
Human's turn again

It's Human's turn!
Please insert a pit number ([1,6]) :->    2
Human moved from 2


_____________________________________________

      4     4     4     4     4     4                Computer
0                                         1
      4     0     1     6     6     6          -->> Human
_____________________________________________
Computer moved from 4


_____________________________________________

      5     5     5     0     4     4          -->> Computer
1                                         1
      4     0     1     6     6     6                Human
_____________________________________________
Computer's turn again
Computer moved from 1
```

Fig: Human and computers are taking turns according to the rule and playing game

```
_____________________________________

      1    7    7    2    6    0          -->> Computer
7                                      2
      0    1    2    7    6    0               Human
_____________________________________

It's Human's turn!
Please insert a pit number ([1,6]) :->   1
Human moved from 1
Invalid!
You cannot select Pits without any stones in it (Pits=0). So please choose again.
```

Fig: Showing error in case of trying to take stone from empty pit

```
_____________________________________

      1    7    7    2    6    0               Computer
7                                      2
      0    1    2    7    6    0          -->> Human
_____________________________________
Human's turn again

It's Human's turn!
Please insert a pit number ([1,6]) :->   8
ERROR: Invalid Input
Please select between ([1,6]) :->   a
Only integer between ! to 6 is valid input.
Please insert a pit number ([1,6]) :->
```

Fig: Showing error in case of input out of range or non-integer value

```
********** Game Over**********
Human scores 20
Computer scores 25
Computer wins
```

Fig: Final result

Discussion: Basically, we divided the whole problem into 4 classes: Driver class, Board class, getPit class, agent class. Driver class was the main class. Basic calculations of Pits and stones were done in getPit class. Whole board implementations, valid moves and game over or not related rules were implemented in the Board class. Finally, the Agent class is divided into two part. For human agent, it will take user input and work accordingly. For machine agent, it will use Alpha-beta pruning algorithm to take best possible move.

In this project, we gained experience of designing a program and got to know how important it is to get a structure. While implementing Alpha-beta pruning algorithm, it was a challenge to consider the case where if the last piece a player drops is in an empty hole on his side, then he will capture that piece and any pieces in the hole directly opposite. While implementing human agent, we struggled a bit to check non integer user input as it was giving error for a long time. However, we were able to solve it successfully.

Conclusion: The Mancala game is a famous adversarial search-based problem. This classic strategy game requires fair amount of foresight and implementing alpha-beta pruning algorithm was quite challenging. We have achieved correct output so far. So, we can say that our game designing was successful. This experiment can help us to implement AI adversarial-techniques to implement more competitive games in future.