Before:

if ((i - states.Count) % 2 == 0){

float value = -((states.Count - (states.Count - i) % 2)) / (float)states.Count;

values.Add(value);

}else{

float value = ((states.Count - (states.Count - i) % 2)) / (float)states.Count;

values.Add(value);

}

After:

if ((i - states.Count) % 2 == 0){

values.Add(-((states.Count - (states.Count - i) % 2)) / (float)states.Count);

}else{

values.Add(((states.Count - (states.Count - i) % 2)) / (float)states.Count);

}

Change: Limited variable scope.

Reason: value was only needed for one purpose, and only that purpose, it was unnecessary to store it in the first place.

Before:

Console.WriteLine("Choose number of row");

Console.WriteLine("(0) for First Row, (1) for Second Row, (2) for Third Row");

string rowNumber = Console.ReadLine();

int row = int.Parse(rowNumber);

Console.WriteLine("How many pieces do you want to take away?");

string takePieces = Console.ReadLine();

int num = int.Parse(takePieces);

After:

Console.WriteLine("Choose number of row");

Console.WriteLine("(0) for First Row, (1) for Second Row, (2) for Third Row");

int row = int.Parse(Console.ReadLine());

Console.WriteLine("How many pieces do you want to take away?");

int num = int.Parse(Console.ReadLine());

Change: Limited variable scope.

Reason: Same as previous change, storing the strings rowNumber and takePieces was unnecessary, as they are only used once.

Before:

HumanPlayer p1 = new HumanPlayer();

ComputerPlayer p2 = new ComputerPlayer();

After:

HumanPlayer player1 = new HumanPlayer();

ComputerPlayer computerPlayer = new ComputerPlayer();

Change: Changed the name of the player variables.

Reason: This makes it more clear what the variables are and do.

Before:

public void endGame()

{

for (int k = 0; k < game.TotalStates; k++)

{

for (int i = 0; i < stateList.Count; i++)

{

if ( stateList[i].TopRow == game.TopRow[k] &&

stateList[i].MidRow == game.MidRow[k] &&

stateList[i].BotRow == game.BotRow[k];

{

stateList[i].addInstance(game.Values[k]);

}

}

}

}

After:

public void endGame()

{

for (int k = 0; k < game.TotalStates; k++)

{

for (int i = 0; i < stateList.Count; i++)

{

if (statesEqualGame(i, k))

{

stateList[i].addInstance(game.Values[k]);

}

}

}

}

public bool statesEqualGame(int statesLists, int gameRows)

{

return (stateList[statesLists].TopRow == game.TopRow[gameRows] &&

stateList[statesLists].MidRow == game.MidRow[gameRows] &&

stateList[statesLists].BotRow == game.BotRow[gameRows]);

}

Change: Changed the condition of the if statement to a method that checks if game states equal the current game.

Reason: This way it is more clear what the if statement is check for.

Before:

if (row >= rows.Length || rows[row] < num || num <= 0)

{

After:

if (noPossibleMove( row, num))

{

public bool noPossibleMove(int row, int num)

{

return (row >= rows.Length || rows[row] < num || num <= 0);

}

Change: Made a method that returns if the move if the move is not possible.

Reason: Replaces the Boolean expression with a self-documenting method.

Before:

if (game.stateList[i].TopRow <= game.rows[0] &&

game.stateList[i].MidRow <= game.rows[1] &&

game.stateList[i].BotRow <= game.rows[2])

After:

if (validMove(game, numState))

public bool validMove(GameManager game, int numState)

{

return (game.stateList[numState].TopRow <= game.rows[0] &&

game.stateList[numState].MidRow <= game.rows[1] &&

game.stateList[numState].BotRow <= game.rows[2]);

}

Change: Made a Boolean method that returns if the move is valid

Reason: Replacing compound conditional with a predicate.