```
/*****************************************************************************
Title:          Connect 3.1 - Game.cs
Name:           Jordan Hanson
Course:         CSCI299.02
Instructor:     Jordan Ringenberg
Last Editted:   December 13, 2015

Description:    This class defines the variables and functions used by the
                AI player. The AI player uses the minimax algorithm to, based on
                a set recursion depth (which corresponds to the "difficulty"
                level of the AI Player), automatically select a strategic
                position on the Connect 3.1 game board.

Input:          Positions currently taken on the gameboard. AI difficulty.

Output:         Where the AI player chooses to play.
*****************************************************************************/
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Connect31_Jordan_Hanson
{
    class Spot
    {
        public int row;
        public int col;
        public int score;
        public Spot()
        {
            row = -1;
            col = -1;
            score = 0;
        }
    }
    class AI_Player
    {
        public Spot GetBestMove(Game G, double difficulty)
        {
            Spot s = Minimax(G, 0, difficulty, G.getTurn());
            return s;
        }
        private Spot Minimax(Game G, int level, double maxLevel, char turn)
        {
            Spot BestSpot = new Spot();

            //if current AI player
            if (level % 2 == 0)
            {
                BestSpot.score = -1000000; //-infinite so AI can maximize the score
            }
```

```
            //if AI opponent
            else
            {
                BestSpot.score = 1000000;//infinite so AI can minimize the score
            }

            //if there is a winner or we are as deep as we are allowed to go
            if (G.checkWinner().Item1 != ' ' || G.BoardIsFull() || level >= maxLevel)
            {
                BestSpot.score = scoreGame(G, level + 1, turn);
                //MessageBox.Show(string.Format("Score is {0}, level is {1}", BestSpot.score,
                level));
            }
            else
            {
                //note that consts in C# are static, meaning that to access
                //them you should use the class name
                for (int i = 0; i < Game.ROWS; i++)
                {
                    for (int j = 0; j < Game.COLS; j++)
                    {
                        if (G.CellIsPlayable(i, j))
                        {
                            Game TmpGame = new Game(G);

                            TmpGame.MarkGameBoard(i, j);

                            Spot TmpSpot = new Spot();
                            TmpSpot.row = i;
                            TmpSpot.col = j;

                            //if (TmpGame.checkWinner().Item1 != ' ')
                            //{
                            TmpSpot.score = Minimax(TmpGame, level + 1, maxLevel, turn).score;

                            //MessageBox.Show(string.Format("Score is {0}, level is {1}",
                            TmpSpot.score, level));
                            //if current AI player
                            if (level % 2 == 0)
                            {
                                if (TmpSpot.score > BestSpot.score)
                                {
                                    BestSpot = TmpSpot;
                                }
                            }
                            //if AI opponent
                            else
                            {
                                if (TmpSpot.score < BestSpot.score)
                                {
                                    BestSpot = TmpSpot;
                                }
                            }
```

```
                    //}
                }
            }
        }
    }
    //MessageBox.Show(string.Format("Score is {0}, level is {1}", BestSpot.score,
    level));
    return BestSpot;
}

/*****************************************************************************
This function combines all the scores of possible moves to determine the best
move to make.
*****************************************************************************/
private int scoreGame(Game G, int level, char turn)
{
    int score = 0;

    score += ScoreWin(G, turn) / (level * level);
    score += ScoreThreeInRow(G, turn) / (level * level);
    score += ScoreL(G, turn) / (level * level);
    score += ScoreTwoInRowOpen(G, turn) / (level * level);

    return score;
}

/*****************************************************************************
This function scores the current move based if the opponent or AI will win.
*****************************************************************************/
private int ScoreWin(Game G, char turn)
{
    int score = 0;
    char checkWinner = G.checkWinner().Item1;
    if (checkWinner != ' ')
    {
        if (checkWinner == turn)
        {
            score += 100000;
        }
        else
        {
            score -= 100000;
        }
    }
    return score;
}
```

```
        /*****************************************************************************
        This function scores the current move based on opponent or AI's placements to
        acquire three in a row on the board.
        *****************************************************************************/
        private int ScoreThreeInRow(Game G, char turn)
        {
            int score = 0;
            //check horizontal
            for (int j = 0; j < Game.COLS - 2; j++)
            {
                for (int i = Game.ROWS - 2; i > -1; i--)
                {
                    if ((G.getGameBoard()[i + 1, j] != ' ' && G.getGameBoard()[i + 1, j] == G.
                    getGameBoard()[i + 1, j + 1])
                        && (G.getGameBoard()[i + 1, j + 1] != ' ' && G.getGameBoard()[i + 1, j +
                         1] == G.getGameBoard()[i + 1, j + 2])
                        && G.getGameBoard()[i + 1, j + 2] != ' ' && (G.CellIsPlayable(i, j) || G
                        .CellIsPlayable(i, j + 2)))
                    {
                        if ((G.getGameBoard()[i + 1, j] == turn && G.getGameBoard()[i + 1, j + 1
                        ] == turn)
                            || (G.getGameBoard()[i + 1, j + 1] == turn && G.getGameBoard()[i + 1
                            , j + 2] == turn))
                            //(G.getGameBoard()[i + 1, j] == turn || G.getGameBoard()[i + 1, j
                            + 1] == turn || G.getGameBoard()[i + 1, j + 2] == turn)
                        {
                            score += 1000;
                        }
                        else
                        {
                            score -= 1000;
                        }
                    }
                    if ((G.getGameBoard()[i, j] != ' ' && G.getGameBoard()[i, j] == G.
                    getGameBoard()[i, j + 1])
                        && (G.getGameBoard()[i, j + 1] != ' ' && G.getGameBoard()[i, j + 1] == G
                        .getGameBoard()[i, j + 2])
                        && G.getGameBoard()[i, j + 2] != ' ' && (G.CellIsPlayable(i + 1, j) || G
                        .CellIsPlayable(i + 1, j + 2)))
                    {
                        if ((G.getGameBoard()[i, j] == turn && G.getGameBoard()[i, j + 1] ==
                        turn)
                            || (G.getGameBoard()[i, j + 1] == turn && G.getGameBoard()[i, j + 2]
                             == turn))
                        {
                            score += 1000;
                        }
                        else
                        {
                            score -= 1000;
                        }
                    }
                }
```

```
        }

        //check vertical
        for (int j = 0; j < Game.COLS - 1; j++)
        {
            for (int i = Game.ROWS - 1; i > 1; i--)
            {
                if ((G.getGameBoard()[i, j] != ' ' && G.getGameBoard()[i, j] == G.
                getGameBoard()[i - 1, j])
                    && (G.getGameBoard()[i - 1, j] != ' ' && G.getGameBoard()[i - 1, j + 1]
                    == G.getGameBoard()[i - 2, j])
                    && G.getGameBoard()[i - 2, j] != ' ' && (G.CellIsPlayable(i, j + 1) || G
                    .CellIsPlayable(i - 2, j + 1)))
                {
                    if ((G.getGameBoard()[i, j] == turn && G.getGameBoard()[i - 1, j] ==
                    turn)
                        || (G.getGameBoard()[i - 1, j] == turn && G.getGameBoard()[i - 2, j]
                         == turn))
                        //(G.getGameBoard()[i, j] == turn || G.getGameBoard()[i - 1, j] ==
                        turn || G.getGameBoard()[i - 2, j] == turn)
                    {
                        score += 1000;
                    }
                    else
                    {
                        score -= 1000;
                    }
                }
                if ((G.getGameBoard()[i, j + 1] != ' ' && G.getGameBoard()[i, j + 1] == G.
                getGameBoard()[i - 1, j + 1])
                    && (G.getGameBoard()[i - 1, j + 1] != ' ' && G.getGameBoard()[i - 1, j +
                     1] == G.getGameBoard()[i - 2, j + 1])
                    && G.getGameBoard()[i - 2, j + 1] != ' ' && (G.CellIsPlayable(i, j) || G
                    .CellIsPlayable(i - 2, j)))
                {

                    if ((G.getGameBoard()[i, j + 1] == turn && G.getGameBoard()[i - 1, j + 1
                    ] == turn)
                        || (G.getGameBoard()[i - 1, j + 1] == turn && G.getGameBoard()[i - 2
                        , j + 1] == turn))
                        //(G.getGameBoard()[i, j + 1] == turn || G.getGameBoard()[i - 1, j
                        + 1] == turn || G.getGameBoard()[i - 2, j + 1] == turn)
                    {
                        score += 1000;
                    }
                    else
                    {
                        score -= 1000;
                    }
                }
            }
        }
        return score;
```

```
    }

    /*************************************************************************
    This function scores the current move based on opponent or AI's placements to
    acquire an L shape on the board and having a playable space to win next to it.
    *************************************************************************/
    private int ScoreL(Game G, char turn)
    {
        int score = 0;
        //check L
        for (int i = Game.ROWS - 1; i > 1; i--)
        {
            for (int j = 1; j < Game.COLS - 1; j++)
            {
                if ((G.getGameBoard()[i, j] != ' ' && G.getGameBoard()[i, j] == G.
                getGameBoard()[i, j + 1])
                    && (G.getGameBoard()[i, j + 1] != ' ' && G.getGameBoard()[i, j + 1] == G
                    .getGameBoard()[i - 1, j + 1])
                    && G.getGameBoard()[i - 1, j + 1] != ' ' && (G.CellIsPlayable(i, j - 1)
                    || G.CellIsPlayable(i - 2, j + 1)))
                {
                    if ((G.getGameBoard()[i, j] == turn && G.getGameBoard()[i, j + 1] ==
                    turn)
                        || (G.getGameBoard()[i, j + 1] == turn && G.getGameBoard()[i - 1, j
                        + 1] == turn))
                        //(G.getGameBoard()[i, j] == turn || G.getGameBoard()[i, j + 1] ==
                        turn || G.getGameBoard()[i - 1, j + 1] == turn)
                    {
                        score += 500;
                    }
                    else
                    {
                        score -= 500;
                    }
                }
            }
            for (int j = 0; j < Game.COLS - 2; j++)
            {
                if ((G.getGameBoard()[i, j] != ' ' && G.getGameBoard()[i, j] == G.
                getGameBoard()[i - 1, j])
                    && (G.getGameBoard()[i - 1, j] != ' ' && G.getGameBoard()[i - 1, j] == G
                    .getGameBoard()[i, j + 1])
                    && G.getGameBoard()[i, j + 1] != ' ' && (G.CellIsPlayable(i - 2, j) || G
                    .CellIsPlayable(i, j + 2)))
                {
                    if ((G.getGameBoard()[i, j] == turn && G.getGameBoard()[i - 1, j] ==
                    turn)
                        || (G.getGameBoard()[i, j] == turn && G.getGameBoard()[i, j + 1] ==
                        turn))
                        //(G.getGameBoard()[i, j] == turn || G.getGameBoard()[i - 1, j] ==
                        turn || G.getGameBoard()[i, j + 1] == turn)
                    {
                        score += 500;
```

```
                }
                else
                {
                    score -= 500;
                }
            }
        }
    }
    for (int i = Game.ROWS - 2; i > 0; i--)
    {
        for (int j = 1; j < Game.COLS - 2; j++)
        {
            if ((G.getGameBoard()[i - 1, j] != ' ' && G.getGameBoard()[i - 1, j] == G.
            getGameBoard()[i - 1, j + 1])
                && (G.getGameBoard()[i - 1, j + 1] != ' ' && G.getGameBoard()[i - 1, j +
                 1] == G.getGameBoard()[i, j + 1])
                && G.getGameBoard()[i, j + 2] != ' ' && (G.CellIsPlayable(i - 1, j - 1)
                || G.CellIsPlayable(i + 1, j + 1)))
            {
                if ((G.getGameBoard()[i - 1, j] == turn && G.getGameBoard()[i - 1, j + 1
                ] == turn)
                    || (G.getGameBoard()[i - 1, j + 1] == turn && G.getGameBoard()[i, j
                    + 1] == turn))
                    //(G.getGameBoard()[i - 1, j] == turn || G.getGameBoard()[i - 1, j
                    + 1] == turn || G.getGameBoard()[i, j + 1] == turn)
                {
                    score += 500;
                }
                else
                {
                    score -= 500;
                }
            }
        }
        for (int j = 0; j < Game.COLS - 2; j++)
        {
            if ((G.getGameBoard()[i + 1, j] != ' ' && G.getGameBoard()[i + 1, j] == G.
            getGameBoard()[i - 1, j])
                && (G.getGameBoard()[i - 1, j] != ' ' && G.getGameBoard()[i - 1, j] == G
                .getGameBoard()[i - 1, j + 1])
                && G.getGameBoard()[i - 1, j + 1] != ' ' && (G.CellIsPlayable(i + 1, j)
                || G.CellIsPlayable(i - 1, j + 2)))
            {
                if ((G.getGameBoard()[i, j] == turn && G.getGameBoard()[i - 1, j] ==
                turn)
                    || (G.getGameBoard()[i - 1, j] == turn && G.getGameBoard()[i - 1, j
                    + 1] == turn))
                    //(G.getGameBoard()[i, j] == turn || G.getGameBoard()[i - 1, j] ==
                    turn || G.getGameBoard()[i - 1, j + 1] == turn)
                {
                    score += 500;
                }
                else
```

```
                {
                    score -= 500;
                }
            }
        }
    }
    return score;
}

/*****************************************************************************
This function scores the current move based on opponent or AI's placements to
acquire two in a row on the board.
*****************************************************************************/
private int ScoreTwoInRowOpen(Game G, char turn)
{
    int score = 0;
    //check horizontal
    for (int j = 0; j < Game.COLS-2; j++)
    {
        for (int i = Game.ROWS - 1; i > -1; i--)
        {
            if ((G.getGameBoard()[i, j] != ' ' && G.getGameBoard()[i, j + 1] != ' '
                 && G.getGameBoard()[i, j] == G.getGameBoard()[i, j + 1] && G.
                 CellIsPlayable(i, j + 2))
                || (G.getGameBoard()[i, j + 1] != ' ' && G.getGameBoard()[i, j + 2] !=
                ' '
                    && G.getGameBoard()[i, j + 1] == G.getGameBoard()[i, j + 2] && G.
                    CellIsPlayable(i, j))
                || (G.getGameBoard()[i, j] != ' ' && G.getGameBoard()[i, j + 2] != ' '
                    && G.getGameBoard()[i, j] == G.getGameBoard()[i, j + 2] && G.
                    CellIsPlayable(i, j + 1)))
            {
                if (G.getGameBoard()[i, j] == turn || G.getGameBoard()[i, j + 1] == turn)
                {
                    score += 100;
                }
                else
                {
                    score -= 100;
                }
            }
        }
    }

    //check vertical
    for (int j = 0; j < Game.COLS; j++)
    {
        for (int i = Game.ROWS - 1; i > 1; i--)
        {
            if ((G.getGameBoard()[i, j] != ' ' && G.getGameBoard()[i - 1, j] != ' '
                 && G.getGameBoard()[i, j] == G.getGameBoard()[i - 1, j] && G.
                 CellIsPlayable(i - 2, j))
                || (G.getGameBoard()[i - 1, j] != ' ' && G.getGameBoard()[i - 2, j] !=
```

```
                        ' '
                            && G.getGameBoard()[i - 1, j] == G.getGameBoard()[i - 2, j] && G.
                            CellIsPlayable(i, j))
                        || (G.getGameBoard()[i, j] != ' ' && G.getGameBoard()[i - 2, j] != ' '
                            && G.getGameBoard()[i, j] == G.getGameBoard()[i - 2, j] && G.
                            CellIsPlayable(i - 1, j)))
                    {

                        if (G.getGameBoard()[i, j] == turn || G.getGameBoard()[i - 1, j] == turn)
                        {
                            score += 100;
                        }
                        else
                        {
                            score -= 100;
                        }
                    }
                }
            }
            return score;
        }
    }
}
```