AI for a Connect 4 Game

Michele Shock
December 6, 2007
Dr. Adnan Shaout
Electrical and Computer Engineering Department
University of Michigan – Dearborn

***Abstract*—This paper looks at implementing artificial intelligence, AI, into an existing Connect 4 game. The AI for this game is based on influence mapping.**

***Key Words*—Real Time Systems, Games, Graphical User Interfaces, Programming, Software Requirements and Specifications, Documentation, Artificial Intelligence**

I. Introduction

Connect 4 was first published by Milton Bradley in 1974.[1] The waterfall model was used in the creation of the software [8]. The concept of the Connect 4 game is to get four chips in a row either diagonally, vertically and horizontally before your opponent. Different AI techniques were studied and an algorithm was chosen.

There are many ways to solve the Connect 4 game. There are several levels of AI difficulty as seen in [5]. They are random, defensive and aggressive AI. Random is just like it sounds; it randomly picks a play and is the easiest to beat. Defensive AI makes blocking a win a priority, while aggressive makes winning a priority. Both are harder to beat than the random AI.

Solution algorithms for AI are numerous and can be complex. Some that were considered are minimax, minimax with alpha-beta pruning, A* and influence maps. Minimax is a recursive tree that uses backtracking to find the optimal move and is the hardest to beat [3][4][5]. The opponents are referred to as MIN and MAX. MIN tries to minimize MAX's score and MAX tries to maximize his score [6]. The algorithm then takes this and looks several moves ahead for the best move possible. Minimax may be the best way of getting the optimal move, but it takes a lot of processing, so pruning methods are used [3]. The pruning method that was considered is the alpha-beta method. Since minimax looks at all possible plays including ones that can be ignored, alpha-beta pruning is used to improve the efficiency of the minimax algorithm [6]. It scores the possible plays and if it is at a MAX node, it only looks down the branches that have a score greater than or equal to that of the MAX node and if it's a MIN node, it looks for a score that is less than or equal to the MIN node [6]. Even with these pruning methods found in [3][4][6], this type of AI is too complex for the existing program. The A* algorithm was also looked at. A* is a best first search that combines the path cost from the start to the end and the estimated cost of the cheapest path [7]. This method would not be a good fit for the Connect 4 game.

The algorithm used in this program is based on using heuristics with influence mapping that is seen in [2]. This was chosen because it is the method that fits in with the original code for the game. It is a greedy algorithm and may not return the optimal move, but it meets the requirements of the system without too much overhead. In [2] there are influence maps and heuristics of Tic-Tac-Toe and Pente, a larger scale Tic-Tac-Toe. Its method looks at and evaluates the entire board, where in Connect 4 only the top available spot in each column needs to be looked at. This program's algorithm uses the basics from [2], transforms them into use for Connect 4 and integrates the aggressive, defensive and random AI from [5].

II. Previous Work

A. Background

This project was originally done as a real-time project for Embedded Systems. The program was written in C using LabWindows/CVI by National Instruments. It was written for a windows environment. The game was 2 players and had a five second timer for play. The plays were made on an interactive GUI. The current player was displayed on the top of the GUI. Once the current player makes a play, the chip is placed in that column within one second. When the chip is in place, the timer is reset. When the timer expires, the current player lost their turn. Due to the time constraints the computer player was not implemented.

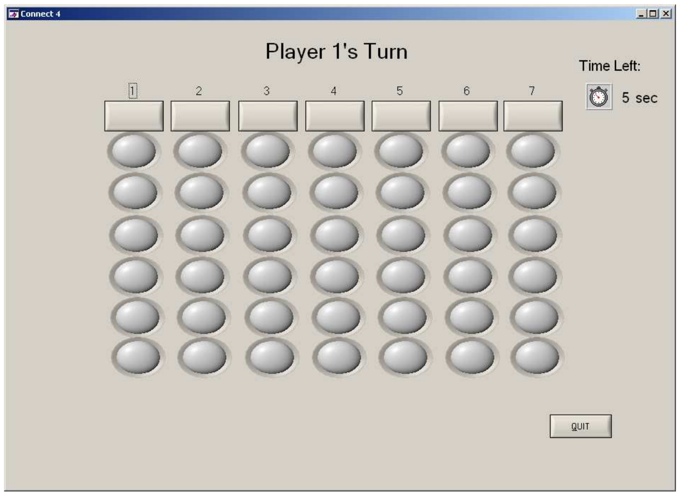


Figure 1: Board Layout

B. *Original Design*

The design is a grid of six rows by seven columns. [1] This layout is seen in Figure 1 and the original design flow is seen in Figure 2. When it is the player's turn to play, they will click on one of the boxes above the circle LEDs to select the column for chip placement. The chip will then be placed at the bottom of the chosen column. After each play, the program looks to see if there is a winner or a tie.

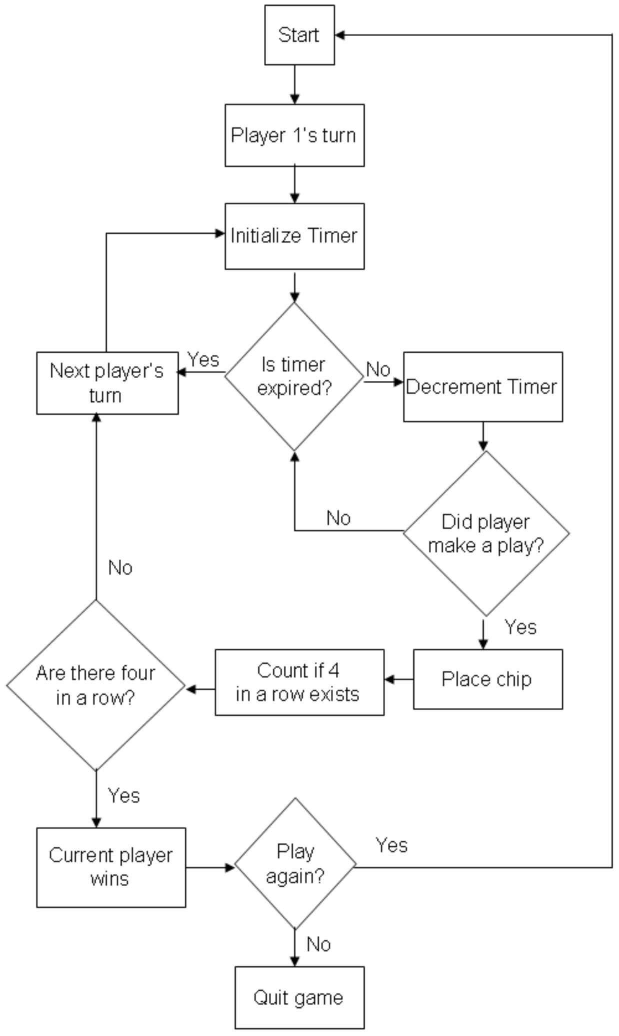


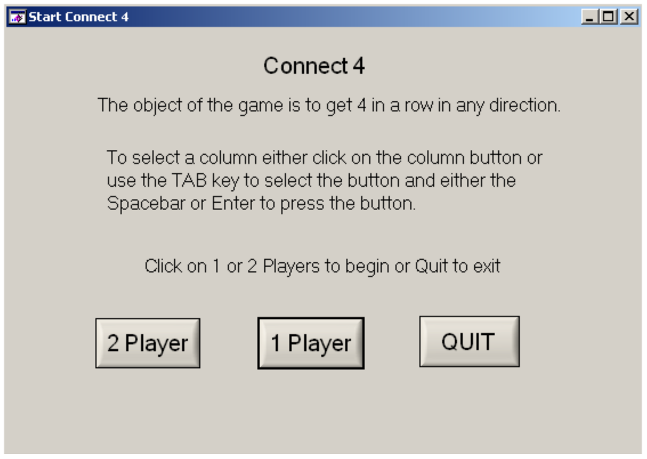
Figure 2: Original Software Design Flow Chart

III. Requirements

The main requirement for this project is that the artificial intelligence is implemented into the original project. It would use the original GUI and may or may not keep the timer depending on algorithm. The program would offer the choice of one or two players.

IV. Software Design

The program starts with the screen seen in Figure 3. This screen gives the player(s) a choice of one player, two players or to quit the game. Figure 4 shows this and shows where the program continues from there.


Start Connect 4

Connect 4

The object of the game is to get 4 in a row in any direction.

To select a column either click on the column button or use the TAB key to select the button and either the Spacebar or Enter to press the button.

Click on 1 or 2 Players to begin or Quit to exit

Figure 3: Start Screen

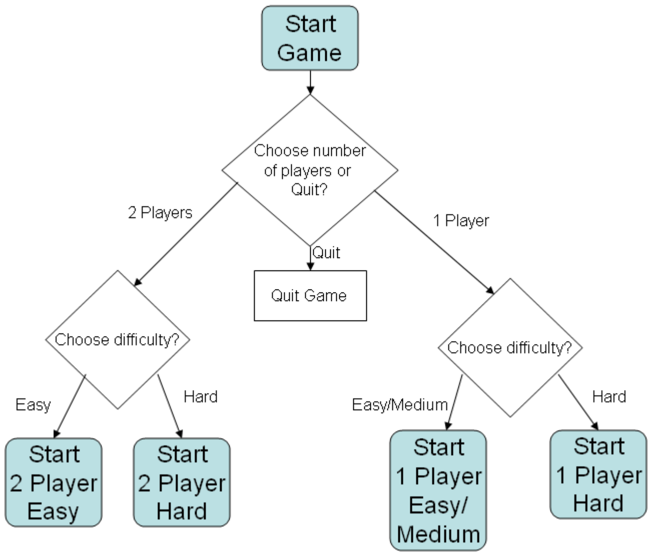
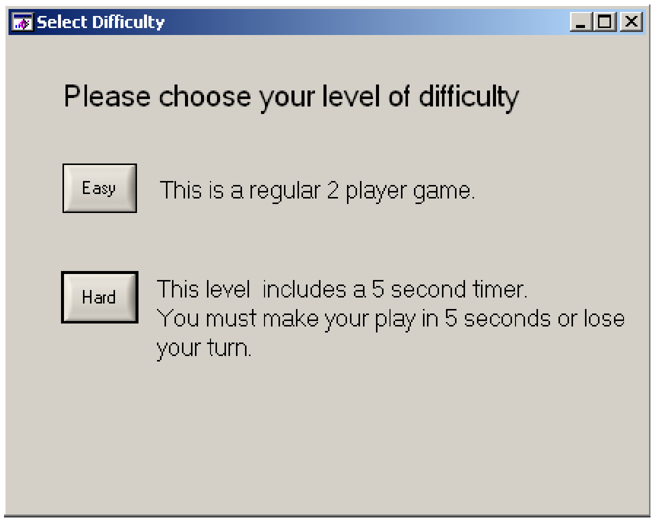


Figure 4: Software Flow of Start and Difficulty Screens


Select Difficulty

Please choose your level of difficulty

This is a regular 2 player game.

This level includes a 5 second timer. You must make your play in 5 seconds or lose your turn.

Figure 5: Two Player Difficulty Screen

The two player game was implemented in the original program. This update allows the players to choose to have a timer or to not have the timer. Figure 5 shows the difficulty choice screen and Figures 6 and 7 show the flow of the difficulty choices. The easy level has no timer and allows for a more leisurely game. In this version the timer will be invisible and disabled. It will wait until the current player makes a move or selects quit. After a play is made, it looks to see if that player has won or caused a tie. The hard level initiates the timer and makes it visible to the players. It plays the same as the two player easy mode except for the lost turn when the timer expires.

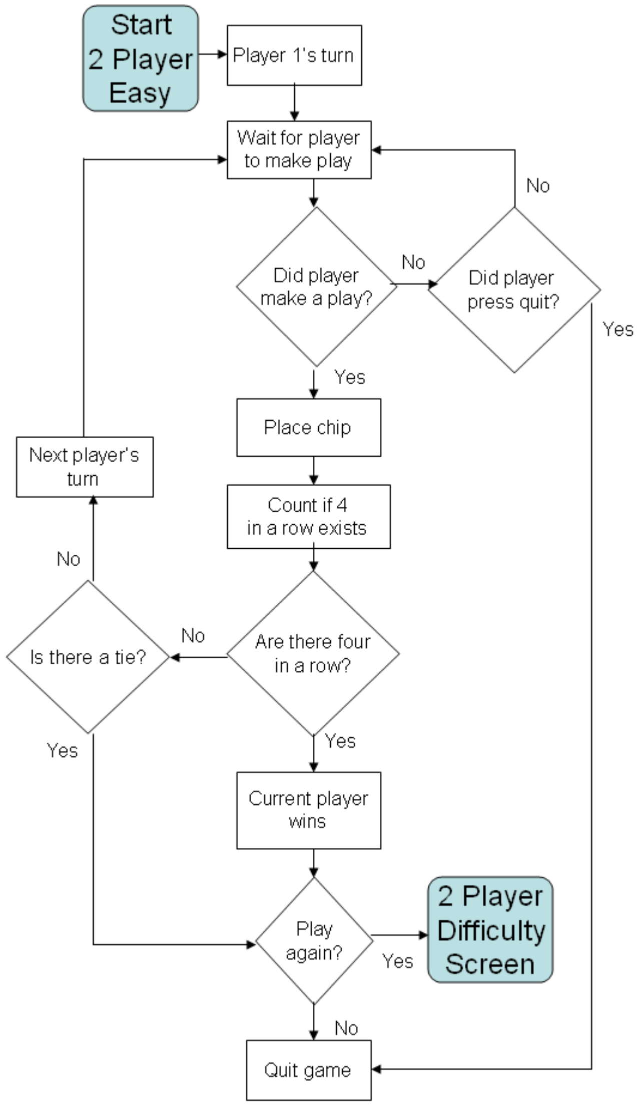


Figure 6: Software Flow for Two Player Easy Mode

The one player game has three levels to choose from. The difficulty screen for the one player option is seen in Figure 8. It allows the player to choose easy, medium or hard. Easy has an AI that is easier to beat. Medium includes the AI from the easy level and adds to it making it harder to beat. Hard has the same AI that is in the Medium and adds the timer to increase the skill level needed to beat the computer.

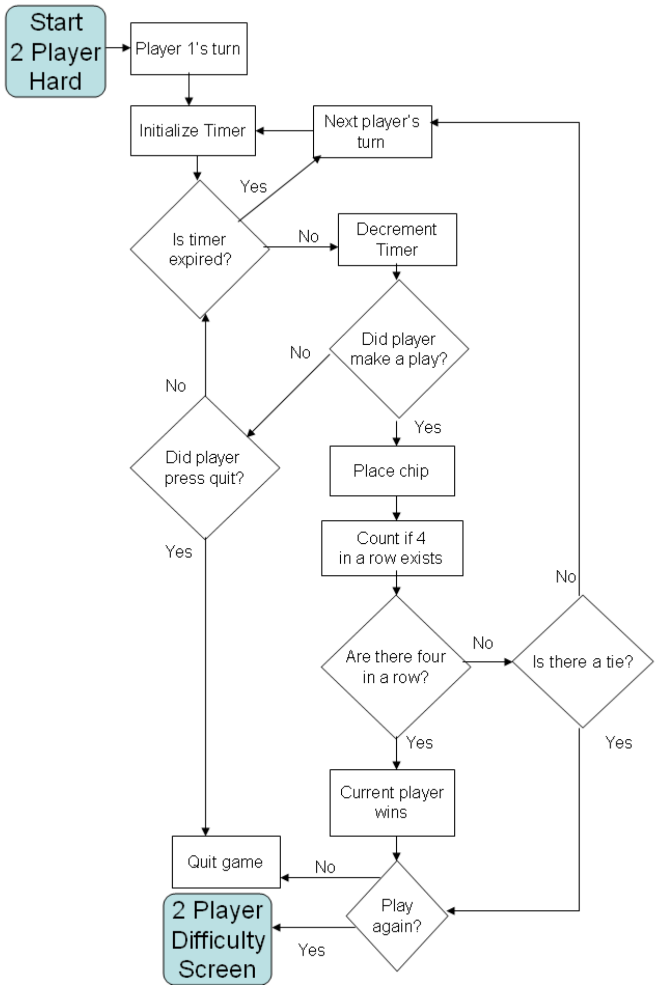
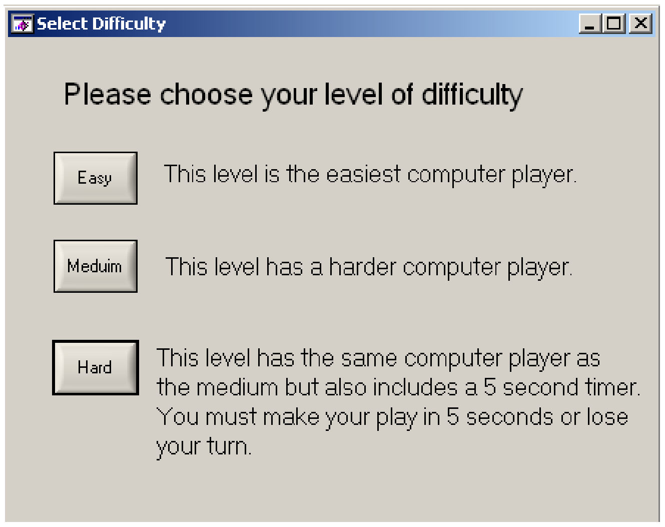


Figure 7: Software Flow for Two Player Hard Mode



Please choose your level of difficulty

This level is the easiest computer player.

This level has a harder computer player.

This level has the same computer player as the medium but also includes a 5 second timer. You must make your play in 5 seconds or lose your turn.

Figure 8: One Player Difficulty Screen

All of the difficulty levels have the same algorithm to score the possible moves. There are only seven possible moves, the bottom spot available for each column. The score is stored in a double array and it keeps track of both the computer player's score and the human player's score for each available move. It starts by looking at the horizontal direction for the move it is scoring. It examines the space to the left of the move (as long as the spot is on the board, it does not check spots that are off the board) and sees if the player it is scoring has a chip there. If that player has a chip there, the count is increased. It continues to count until either the space is not occupied for the player being scored or it reaches the end of the board. It then looks to the right and does the same thing. For example, while scoring the computer player, the move it is examining has a chip to the left that is the computers and a chip to the right that is the human players. It sees the chip on the left and increases the count to 1. It then looks another space to the left and sees it empty and stops looking left. It then looks right and sees that the other player has a chip there, so it stops looking to the right. The score for this move is 1.

Scoring is then done in the vertical direction. Each possible move looks down until it hits the other players chip or the bottom of the board. If the score for this column is higher than the score for the horizontal direction, the horizontal score is replaced with the vertical score. The moves are then scored in the left and right diagonal directions. The scoring algorithm checks down and up the diagonal for the number of chips in a row that are connected to that move until they hit the other players chip, an empty space or the end of the board.

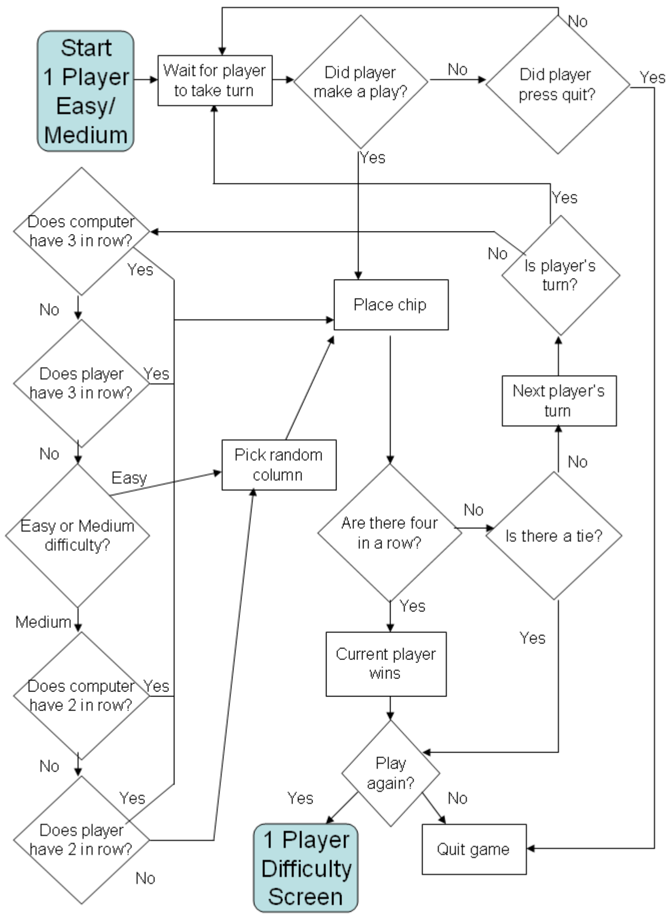


Figure 9: Software Flow for One Player Easy/Medium Mode

The AI for the easy level starts with an aggressive move. It takes the scores for the computer and sees if there is a score of 3 or more, which would be a win the game. If there is a score of 3, it picks that column. If there is no move, it makes a defensive move by looking at the score of the player to see if they have a score of 3 or more. If neither the computer nor the player has a score of 3 or more, then a random column is selected.

The medium level is built off the easy level. Like the easy level it first looks for the computer or the player to have a score of 3 or more. If a move is not found yet, the AI will make another aggressive move by looking to see if the computer has a score of 2 anywhere. This will allow the computer to get closer to a win and force the player to block instead of working toward a win. If that move does not exist, it will then play another defensive move by blocking 2 in a row that the player has. Finally, if neither player has 2 in a row, a random column is chosen. Figure 9 shows the software flow for the easy and the medium levels.

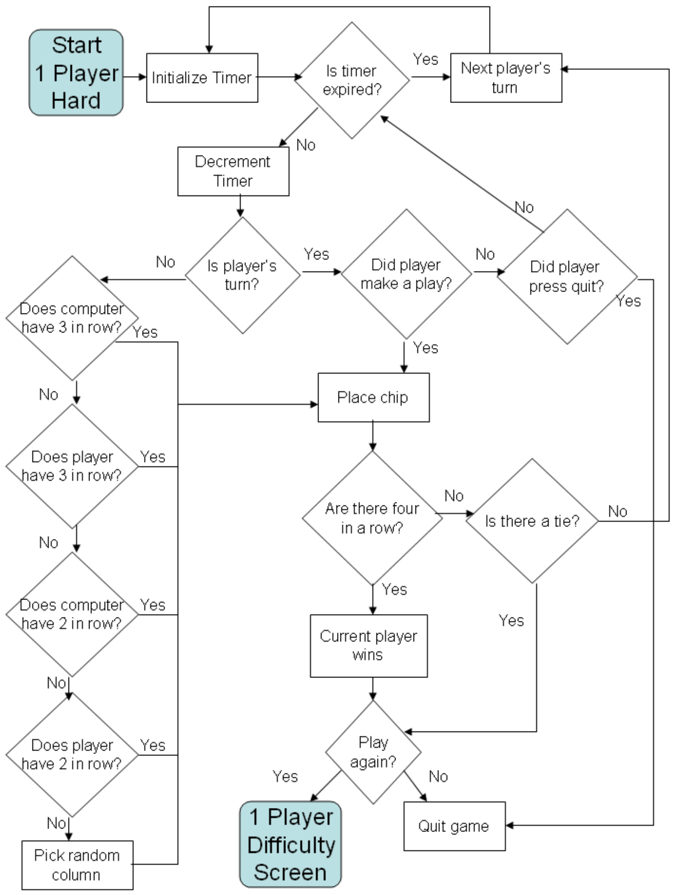


Figure 10: Software Flow for One Player Hard Mode

Picking the hard level gives you the same AI that the medium level has. What makes it harder is that a five second timer is added. The player has five seconds to make a play after the computer plays. This requires quick thinking in order to beat the computer. A delay was added for the computer player so that the move could be seen easily by the human player. Figure 10 shows the software flow for the hard level.

V. Contrast between old and new versions

The difference between the old and the new versions is the choice of a timer and the computer player AI. As shown the timer is made an option where in the old program the timer was always there. There is now a choice between one and two players, where the old version was just a two player game. Implementing these changes required the addition of two new screens and updating the start screen. The start screen now has two buttons that start the game, see Figure 3, instead of one. Figures 5 and 8 are new screens added to implement the difficulty levels and the timer.

VI. Testing

Test procedure consists of playing the game in all the modes.

- Select the 2 player mode.
  - Select easy mode and play the game. Make sure that there are no bugs and that it flows as if you were playing with a live board and chips.
  - Select the hard mode and make sure that it flows like the easy mode except for the timer. Make sure that when the timer expires that the current player changes. Make sure that when the current player makes a move that the next player is made current and the timer resets.
- Select the 1 player mode
  - Select the easy mode, play the game and see where the computer chooses to play. It should try to win or block three in a row, but randomly pick any other play. Run with the debugger and make sure that it never gets to part of the AI that is not intended for it.
  - Select the medium mode, play the game and see where the computer chooses to play. It should try to win or block three in a row, create three in a row or block 2 in a row, then choose randomly. Run with the debugger and make sure that it never gets to part of the AI that is not intended for it.
  - Select the hard mode, and run the same test as the medium mode and add in test for the timer. Make sure that when the timer expires that the current player changes. Make sure that when the current player makes a move that the next player is made current and the timer resets.

For this project, there were several black box testers. Their feedback consisted of:

- Add delay to computer player, it is hard to keep track of where the computer plays when it is right after the player makes a play. After delay was added, it was noted that it might be to long.
- When the game is over and the splash screen asks to play again. If the player chooses to play again the game should either re-start the game at the same level, or go back to the screen were it asks for the difficulty level, but it should not go back to the main screen were it has to choose the number of players.

There was also a white box tester. This tester gave the following feedback:

- The game firsts recognizes that player 1 won, then it continued playing even when the player has won. This resulted in two messages, one that says that the computer won the other one that player 1 won

These items were then investigated, corrected if needed and retested. The testers found no further bugs that needed to be resolved.

VII. Results

The result of this project is that the AI for a Connect 4 game was able to be implemented. It met original requirements and was updated as seen necessary during programming.

VIII. Conclusion

AI for Connect 4 game was integrated into a real-time version using the waterfall software development model. The game was updated to include the choice of one or two players. If two players were chosen, there was a choice of having a timer. If one player was chosen, there was the choice of easy AI, medium AI or hard AI. The only level in the one player mode that has a timer is the hard level. Testing found that the requirements were met. A few things were overlooked and they were found and fixed during testing.

References

[1] http://en.wikipedia.org/wiki/Connect_Four

[2] http://www.generation5.org/content/2000/boardai.asp

[3] http://www.codeproject.com/netcf/Connect4AB.asp

[4] http://www.pomakis.com/c4/connect_generic/c4.txt

[5] http://www.ccs.neu.edu/home/eclip5e/classes/csu520/SmartConnectFour-Final_Paper-v1.0.doc

[6] N. P. Padhy, “State Space Search: Implementation and Applications,” in Artificial Intelligence and Intelligent Systems, New York: Oxford Univ. Press, 2005, ch. 4, sec. 4.10, pp. 161-171

[7] S. J. Russell and P. Norvig, “Informed Search Methods,” in Artificial Intelligence A Modern Approach, Upper Saddle River, NJ: Prentice-Hall, 1995, ch. 4, pp. 96-98

[8] http://en.wikipedia.org/wiki/Waterfall_model

Appendix A

```
/************************************************************

        Connect4.c
        By: Michele Shock
        ECE 554 - Embedded Systems

        Dr. Adnan Shaout
        Electrical and Computer Engineering Department
        University of Michigan - Dearborn

************************************************************/

#include <utility.h>
#include "toolbox.h"
#include <cvirte.h>
#include <userint.h>
#include "Connect 4.h"

//Global variables
const int col = 7;
const int row = 6;
int board[7][8]; //stores player choices
int score[4][8];
int rowindicator[8]; //bottom row available in each column
int columnpick;
int player;  //computer is player 3
int computer;  //is there a computer player? 0 = false, 1 = true
int player1color = VAL_BLUE;
int player2color = VAL_RED;
int computerplayercolor = VAL_WHITE;
int difficulty; //1-easy, 2-medium, 3-hard

static int panel3;//start
static int panel1;//game
static int panel2;//play again?
static int panel4;//difficulty for computer
static int panel5;//difficulty for 2 player

int timerwanted; //0-false, 1-true
int timeelapsed; //timer

void InitGlobal(void);
void pickbestplay (void);
void horizscore (int scoreplayer);
void vertscore (int scoreplayer);
void leftdiagscore (int scoreplayer);
void rightdiagscore (int scoreplayer);

/************************************************************
    InitGlobal()

Initialize global variables that are not constant
************************************************************/
void InitGlobal(void)
{
    int i,j,k;
    //Initialize board to all 0s
    for(i = 1; i <= row; i++)
    {
        for(j = 1; j <= col; j++)
        {
            board[i][j] = 0;
        }
    }

    //Initialize the current row in each column to 1
    for(k = 1; k <= col; k++)
    {
        rowindicator[k] = 1;
    }
    //computer = 0;  //default no computer player
    player = 1;
    timerwanted = 0;
    timeelapsed = 0;
}

/************************************************************
     main()

Start of program
************************************************************/
int main (int argc, char *argv[])
{
    InitGlobal();

    if (InitCVIRTE (0, argv, 0) == 0)
        return -1; /* out of memory */
    if ((panel3 = LoadPanel (0, "Connect 4.uir", PANEL_3)) < 0)
        return -1;
    SetPanelAttribute (panel3, ATTR_TITLE, "Start Connect 4");
    DisplayPanel (panel3);
    RunUserInterface ();
    return 0;
}

/***********************************************************
     pickbestplay()

AI for computer player, allows for the program to pick
column for different difficulty levels
***********************************************************/
void pickbestplay (void)
{
    int bestchoice;
    int realplayer = 1;
    int computerplayer = 3;
    int i,j;
    int found = 0; //false

    Delay (0.5);

    //Initialize score to all 0s
    for(i = 1; i <= 3; i++)
    {
        for(j = 1; j <= col; j++)
        {
            score[i][j] = 0;
        }
    }

    //offensive play to win
    //else if comp has any 3 pick that spot
    horizscore(computerplayer);
    vertscore(computerplayer);
    leftdiagscore(computerplayer);
    rightdiagscore(computerplayer);

    //all difficulty levels

    for(j = 1; j <= col; j++)
    {
        if (score[computerplayer][j] >= 3)
        {
            if (rowindicator[j] <= row)
            {
                bestchoice = j;
                found = 1;
            }
        }
    }


    if (found == 0)
    {
        horizscore(realplayer);
        vertscore(realplayer);
        leftdiagscore(realplayer);
        rightdiagscore(realplayer);

        //defensive play to block player
        //if p1 has any 3, pick that spot
        for(j = 1; j <= col; j++)
        {
            if (score[realplayer][j] >= 3)
            {
                if (rowindicator[j] <= row)
                {
                    bestchoice = j;
                    found = 1;
                }
            }
        }
    }

    if ((found == 0) && (difficulty >= 2))
    {
        //offensive play to create win
        //if computer has any 2, pick that spot
        for(j = 1; j <= col; j++)
        {
            if (score[computerplayer][j] == 2)
            {
                if (rowindicator[j] <= row)
                {
                    bestchoice = j;
                    found = 1;
                }
            }
        }
    }

    if ((found == 0) && (difficulty >= 2))
    {
        //defensive play to create win
        //if player has any 2, pick that spot
        for(j = 1; j <= col; j++)
        {
            if (score[realplayer][j] == 2)
            {
                if (rowindicator[j] <= row)
                {
                    bestchoice = j;
                    found = 1;
                }
            }
        }
    }
    //else pick comp = 2,1,0  but closer to center better choice
    if (found == 0)
    {

        SetRandomSeed (0);
        do
        {
            bestchoice = Random(0,8);
        }while (rowindicator[bestchoice] > row);
    }

    //now that have the column the computer picks, select it
    if (bestchoice == 1)
    {
        Column1 (panel1,2,EVENT_COMMIT, 0, 0, 0);
    }
    else if (bestchoice == 2)
    {
        Column2 (panel1,2,EVENT_COMMIT, 0, 0, 0);
    }
    else if (bestchoice == 3)
    {
        Column3 (panel1,2,EVENT_COMMIT, 0, 0, 0);
    }
    else if (bestchoice == 4)
    {
        Column4 (panel1,2,EVENT_COMMIT, 0, 0, 0);
    }
    else if (bestchoice == 5)
    {
        Column5 (panel1,2,EVENT_COMMIT, 0, 0, 0);
    }
    else if (bestchoice == 6)
    {
        Column6 (panel1,2,EVENT_COMMIT, 0, 0, 0);
    }
    else //if (bestchoice == 7)
    {
        Column7 (panel1,2,EVENT_COMMIT, 0, 0, 0);
    }
}

/************************************************************
     horizscore()

Used only with the computer player, scores the computer and
player in the horizontal direction to find the best move
************************************************************/
void horizscore (int scoreplayer)
{
    int i, checkcol,checkrow;
    int count = 0;
    int done = 0; //false

    for (i = 1; i <= col; i++)
    {
        count = 0;
        checkcol = i;
        checkrow = rowindicator[i];
        done = 0;
        while (checkrow <= 6 && checkcol > 1 && done == 0) //check left
        {
            --checkcol;
            if (board[checkrow][checkcol] == scoreplayer)
                count++;
            else
                done = 1;
        }
        checkcol = i;
        checkrow = rowindicator[i];
        done = 0;
        while (checkrow <= 6 && checkcol < 7 && done == 0) //check right
        {
            ++checkcol;
            if (board[checkrow][checkcol] == scoreplayer)
                count++;
            else
                done = 1;
        }
        if (count > score[scoreplayer][i])
            score[scoreplayer][i] = count;
    }
}

/************************************************************
     vertscore()

Used only with the computer player, scores the computer and
player in the vertical direction to find the best move
************************************************************/
void vertscore (int scoreplayer)
{
    int i, checkrow;
    int count = 0;
    int done = 0; //false

    for (i = 1; i <= col; i++)
    {
        count = 0;
        checkrow = rowindicator[i];
        done = 0;
        while (checkrow > 1 && done == 0) //check down
        {
            --checkrow;
            if (board[checkrow][i] == scoreplayer)
                count++;
            else
                done = 1;
        }
        if (count > score[scoreplayer][i])
            score[scoreplayer][i] = count;
    }
}

/************************************************************
     leftdiagscore()

Used only with the computer player, scores the computer and
player in the left diagonal direction to find the best move
************************************************************/
void leftdiagscore (int scoreplayer)
{
    int i, checkrow, checkcol;
    int count = 0;
    int done = 0; //false

    for (i = 1; i <= col; i++)
    {
        count = 0;
        checkrow = rowindicator[i];
        checkcol = i;
        done = 0;
        while (checkrow > 1 && checkcol < 7 && done == 0) //check down
        {
            --checkrow;
            ++checkcol;
            if (board[checkrow][checkcol] == scoreplayer)
                count++;
            else
                done = 1;
        }
        checkrow = rowindicator[i];
        checkcol = i;
        done = 0;
        while (checkrow < 6 && checkcol > 1 && done == 0) //check up
        {
            ++checkrow;
            --checkcol;
            if (board[checkrow][checkcol] == scoreplayer)
                count++;
            else
                done = 1;
        }
        if (count > score[scoreplayer][i])
            score[scoreplayer][i] = count;
    }
}

/**********************************************************
     rightdiagscore()

Used only with the computer player, scores the computer and
player in the right diagonal direction to find the best move
**********************************************************/
void rightdiagscore (int scoreplayer)
{
    int i, checkrow, checkcol;
    int count = 0;
    int done = 0; //false

    for (i = 1; i <= col; i++)
    {
        count = 0;
        checkrow = rowindicator[i];
        checkcol = i;
        done = 0;
        while (checkrow > 1 && checkcol > 1 && done == 0) //check down
        {
            --checkrow;
            --checkcol;
            if (board[checkrow][checkcol] == scoreplayer)
                count++;
            else
                done = 1;
        }
        checkrow = rowindicator[i];
        checkcol = i;
        done = 0;
        while (checkrow < 6 && checkcol < 7 && done == 0) //check up
        {
            ++checkrow;
            ++checkcol;
            if (board[checkrow][checkcol] == scoreplayer)
                count++;
            else
                done = 1;
        }
        if (count > score[scoreplayer][i])
            score[scoreplayer][i] = count;
    }
}

/**********************************************************
    changeplayer()

Switches the current player
**********************************************************/
void changeplayer(void)
{

     if(computer == 0)
     {
         if(player == 1)
         {
            player = 2;
            SetCtrlVal (panel1, PANEL_TEXTMSG,"Player 2's Turn");
         }
         else
         {
            player = 1;
            SetCtrlVal (panel1, PANEL_TEXTMSG,"Player 1's Turn");
         }
     }
     else
     {
         if(player == 1)
         {
            player = 3;//computer player
            SetCtrlVal (panel1, PANEL_TEXTMSG,"Computer's Turn");
            pickbestplay();
         }
         else
         {
            player = 1;
            SetCtrlVal (panel1, PANEL_TEXTMSG,"Player 1's Turn");
         }
     }
     if (timerwanted == 1)
     {
         ResetTimer (panel1, PANEL_TIMER);
         timeelapsed = 0;
         SetCtrlVal (panel1, PANEL_TEXTMSG_2,"5");
        ResumeTimerCallbacks();
     }
}

/************************************************************
     ishorizwin()
     Inputs:
          rowchoice     current row that is available
          colchoice     column the player has chosen
     Outputs:
          1             winner
          0             no winner

Returns a 1 if there is 4 in a row in the horizontal direction
************************************************************/
int ishorizwin(int rowchoice, int colchoice)
{
     int count = 0;
     int j;
     for(j = 1; j <= col; j++)
     {
          if (board[rowchoice][j] != player)
               count = 0;
          else if (board[rowchoice][j] == player)
          {
               count++;
               if(count == 4)
                    return 1;
          }
     }
     return 0;
}

/************************************************************
     isvertwin()
     Inputs:
          rowchoice     current row that is available
          colchoice     column the player has chosen
     Outputs:
          1             winner
          0             no winner

Returns a 1 if there is 4 in a row in the vertical direction
************************************************************/
int isvertwin (int rowchoice, int colchoice)
{
     int count = 0;
     int i;
     for(i = 1; i <= row; i++)
     {
          if (board[i][colchoice] != player)
               count = 0;
          else if (board[i][colchoice] == player)
          {
               count++;
               if(count == 4)
                    return 1;
          }
     }
     return 0;
}

/************************************************************
     isrightdiagwin()
     Inputs:
          rowchoice     current row that is available
          colchoice     column the player has chosen
     Outputs:
          1             winner
          0             no winner

Returns a 1 if there is 4 in a row in a right slant
diagonal direction
************************************************************/
int isrightdiagwin (int rowchoice, int colchoice)
{
     int startrow,startcol;
     int count;
     int j, i;

     startcol = colchoice;
     startrow = rowchoice;
     count = 0;
     //find lowest spot in diagonal
     while((startrow > 1) && (startcol > 1))
     {
          startrow--;
          startcol--;
     }
```

```
    j = startcol;
    //search upward for 4 in a row
    for(i = startrow; (i <= row) && (j <= col); i++)
    {

        if (board[i][j] != player)
            count = 0;
        else if (board[i][j] == player)
        {
            count++;
            if(count == 4)
                return 1;
        }
        j++;

    }
    return 0;
}

/************************************************************
    isleftdiagwin()
    Inputs:
        rowchoice     current row that is available
        colchoice     column the player has chosen
    Outputs:
        1             winner
        0             no winner

Returns a 1 if there is 4 in a row in a left slant
diagonal direction
************************************************************/
int isleftdiagwin (int rowchoice, int colchoice)
{
    int startrow,startcol;
    int count;
    int j, i;

    startcol = colchoice;
    startrow = rowchoice;
    count = 0;
    //find lowest spot in diagonal
    while((startrow > 1) && (startcol < col))
    {
        startrow--;
        startcol++;
    }
    j = startcol;
    //search upward for 4 in a row
    for(i = startrow; (i <= row) && (j > 1); i++)
    {

        if (board[i][j] != player)
            count = 0;
        else if (board[i][j] == player)
        {
            count++;
            if(count == 4)
                return 1;
        }
        j--;

    }
    return 0;
}

/************************************************************
    iswinner()
    Inputs:
        rowchoice     current row that is available
        colchoice     column the player has chosen
    Outputs:
        1             winner
        0             no winner

Returns a 1 if there is a winner, also suspends the timer
if there is a winner
************************************************************/
int iswinner (int rowchoice, int colchoice)
{
    if (ishorizwin(rowchoice, colchoice) == 1)
    {
        if (timerwanted == 1)
        {
            SuspendTimerCallbacks();
        }
        return 1;
    }
    if (isvertwin(rowchoice, colchoice) == 1)
    {
        if (timerwanted == 1)
        {
            SuspendTimerCallbacks();
        }
        return 1;
    }
    if (isleftdiagwin(rowchoice, colchoice) == 1)
    {
        if (timerwanted == 1)
        {
            SuspendTimerCallbacks();
        }
        return 1;
    }
    if (isrightdiagwin(rowchoice, colchoice) == 1)
    {
        if (timerwanted == 1)
        {
            SuspendTimerCallbacks();
        }
        return 1;
    }
    if (timerwanted == 1)
    {
        SuspendTimerCallbacks();
    }
    return 0;
}

/************************************************************
    istie()
    Outputs:
        1               tie
        0               no tie

Checks the board for any 0s and returns a 1 if there is a tie
************************************************************/
int istie (void)
{
    int i,j;
    for(i = 1; i <= row; i++)
    {
        for(j = 1; j <= col; j++)
        {
            if(board[i][j] == 0)
                return 0;
        }
    }
    return 1;
}


/************************************************************
    loadboard()
    Inputs:
        rowchoice       current row that is available
        colchoice       column the player has chosen
    Outputs:
        1               winner / tie
        0               no winner / tie

Places a 1 for player 1 and a 2 for player 2 in the spot
chosen by the current player
************************************************************/
int loadboard(int columnpick, int rowpick)
{
    int winner,tie;
    int playagain;
    if(rowpick <= row && columnpick <= col)
    {
        board[rowpick][columnpick] = player;
    }
    else
    {
        //column is full
    }
    winner = iswinner (rowpick, columnpick);
    if (winner == 1)//true
    {
        if (player == 1)
        {
            MessagePopup ("WINNER!","Congratulations Player 1 You Win!");
        }
        else if (player == 2)
        {
            MessagePopup ("WINNER!","Congratulations Player 2 You Win!");
        }
        else
        {
            MessagePopup ("WINNER!","Sorry, Computer Wins!");
        }
        if ((panel2 = LoadPanel (0, "Connect 4.uir", PANEL_2)) < 0)
            return -1;
        SetPanelAttribute (panel2, ATTR_TITLE, "");
        DisplayPanel (panel2);
        return 1;
    }
    tie = istie();
    if (tie == 1)//true
    {
        MessagePopup ("TIE!","It's a tie!");
        if ((panel2 = LoadPanel (0, "Connect 4.uir", PANEL_2)) < 0)
            return -1;
        SetPanelAttribute (panel2, ATTR_TITLE, "");
        DisplayPanel (panel2);
        return 1;
    }
    return 0;
}

/************************************************************
    StartProgram()

Called when player choses to start the game with 2 players
from the initial start up panel
************************************************************/
int CVICALLBACK StartProgram (int panel, int control, int event,
        void *callbackData, int eventData1, int eventData2)
{
    switch (event)
        {
        case EVENT_COMMIT:
            computer = 0;
            DiscardPanel (panel3);
            if ((panel5 = LoadPanel (0, "Connect 4.uir", PANEL_5)) < 0)
                return -1;
            SetPanelAttribute (panel5, ATTR_TITLE, "Select Difficulty");
            DisplayPanel (panel5);
            SetActivePanel (panel5);

            break;
        }
    return 0;
}


/************************************************************
    StartComputerProgram()

Called when player choses to start the game with 1 player
from the initial start up panel
************************************************************/
int CVICALLBACK StartComputerProgram (int panel, int control, int event,
        void *callbackData, int eventData1, int eventData2)
{
    switch (event)
        {
        case EVENT_COMMIT:
            computer = 1;
            DiscardPanel (panel3);
            if ((panel4 = LoadPanel (0, "Connect 4.uir", PANEL_4)) < 0)
                return -1;
            SetPanelAttribute (panel4, ATTR_TITLE, "Select Difficulty");
            DisplayPanel (panel4);
            SetActivePanel (panel4);
            break;
        }
    return 0;
}

/************************************************************
    PlayAgain()

Called when player choses to play the game again after there
is a tie or a winner
************************************************************/
int CVICALLBACK PlayAgain (int panel, int control, int event,
        void *callbackData, int eventData1, int eventData2)
{
    switch (event)
        {
        case EVENT_COMMIT:
            DiscardPanel (panel2);
            DiscardPanel (panel1);
            if (computer == 1)
            {
                if ((panel4 = LoadPanel (0, "Connect 4.uir", PANEL_4)) < 0)
                    return -1;
                SetPanelAttribute (panel4, ATTR_TITLE, "Select Difficulty");
                DisplayPanel (panel4);
            }
            else
            {
                if ((panel5 = LoadPanel (0, "Connect 4.uir", PANEL_5)) < 0)
                    return -1;
                SetPanelAttribute (panel5, ATTR_TITLE, "Select Difficulty");
                DisplayPanel (panel5);
            }
            InitGlobal();
            break;
        }

    return 0;
}

/************************************************************
    QuitGame()

Called when player choses to quit the game again after there
is a tie or a winner
************************************************************/
int CVICALLBACK QuitGame (int panel, int control, int event,
        void *callbackData, int eventData1, int eventData2)
{
    switch (event)
        {
        case EVENT_COMMIT:
            DiscardPanel (panel2);
            DiscardPanel (panel1);
            QuitUserInterface (0);
            break;
        }
    return 0;
}

/************************************************************
    TimerExpired()

Called when the timer clicks every second, then will change
the display to decrement the time shown
************************************************************/
int CVICALLBACK TimerExpired (int panel, int control, int event,
        void *callbackData, int eventData1, int eventData2)
{
    switch (event)
        {
        case EVENT_TIMER_TICK:
            if (timerwanted == 1)
            {
                if(timeelapsed == 0)
                {
                    SetCtrlVal (panel1, PANEL_TEXTMSG_2,"5");
                    timeelapsed++;
                }
                else if(timeelapsed == 1)
                {
                    SetCtrlVal (panel1, PANEL_TEXTMSG_2,"4");
                    timeelapsed++;
                }
                else if(timeelapsed == 2)
                {
                    SetCtrlVal (panel1, PANEL_TEXTMSG_2,"3");
                    timeelapsed++;
                }
                else if(timeelapsed == 3)
                {
                    SetCtrlVal (panel1, PANEL_TEXTMSG_2,"2");
                    timeelapsed++;
                }
                else if(timeelapsed == 4)
                {
                    SetCtrlVal (panel1, PANEL_TEXTMSG_2,"1");
                    timeelapsed++;
                }
                else if(timeelapsed == 5)
                {
                    SetCtrlVal (panel1, PANEL_TEXTMSG_2,"0");
                    timeelapsed = 0;
                    changeplayer();
                }
            }

            break;
        }
    return 0;
}
```

```
/************************************************************
     QuitCallback()

Called when the quit button is pressed and ends the game
************************************************************/
int CVICALLBACK QuitCallback (int panel, int control, int event,
          void *callbackData, int eventData1, int eventData2)
{
     switch (event)
          {
          case EVENT_COMMIT:
               QuitUserInterface (0);
               break;
          }
     return 0;
}

/************************************************************
     HardSetting()

Called when the hard setting is chosen in the 1 player mode
************************************************************/
int CVICALLBACK HardSetting (int panel, int control, int event,
          void *callbackData, int eventData1, int eventData2)
{
     switch (event)
          {
          case EVENT_COMMIT:
               difficulty = 3;
               timerwanted = 1;
               if ((panel1 = LoadPanel (0, "Connect 4.uir", PANEL)) < 0)
                    return -1;
               SetPanelAttribute (panel1, ATTR_TITLE, "Connect 4");
               SetCtrlAttribute (panel1, PANEL_TIMER, ATTR_VISIBLE, 1);
               ResetTimer (panel1, PANEL_TIMER);
               DisplayPanel (panel1);
               SetActivePanel (panel1);
               DiscardPanel(panel4);

               break;
          }
     return 0;
}

/************************************************************
     MediumSetting()

Called when the medium setting is chosen in the 1 player mode
************************************************************/
int CVICALLBACK MediumSetting (int panel, int control, int event,
          void *callbackData, int eventData1, int eventData2)
{
     switch (event)
          {
          case EVENT_COMMIT:
               difficulty = 2;
               timerwanted = 0;
               if ((panel1 = LoadPanel (0, "Connect 4.uir", PANEL)) < 0)
                    return -1;
               SetPanelAttribute (panel1, ATTR_TITLE, "Connect 4");
               SetCtrlAttribute (panel1, PANEL_TEXTMSG_2, ATTR_VISIBLE, 0);
               SetCtrlAttribute (panel1, PANEL_TEXTMSG_3, ATTR_VISIBLE, 0);
               SetCtrlAttribute (panel1, PANEL_TEXTMSG_4, ATTR_VISIBLE, 0);
               DisplayPanel (panel1);
               SetActivePanel (panel1);
               DiscardPanel(panel4);

               break;
          }
     return 0;
}

/************************************************************
     EasySetting()

Called when the easy setting is chosen in the 1 player mode
************************************************************/
int CVICALLBACK EasySetting (int panel, int control, int event,
          void *callbackData, int eventData1, int eventData2)
{
     switch (event)
          {
          case EVENT_COMMIT:
               difficulty = 1;
               timerwanted = 0;
               if ((panel1 = LoadPanel (0, "Connect 4.uir", PANEL)) < 0)
                    return -1;
               SetPanelAttribute (panel1, ATTR_TITLE, "Connect 4");
               SetCtrlAttribute (panel1, PANEL_TEXTMSG_2, ATTR_VISIBLE, 0);
```

```
            SetCtrlAttribute (panel1, PANEL_TEXTMSG_3, ATTR_VISIBLE, 0);
            SetCtrlAttribute (panel1, PANEL_TEXTMSG_4, ATTR_VISIBLE, 0);
            DisplayPanel (panel1);
            SetActivePanel (panel1);
            DiscardPanel(panel4);

            break;
        }
    return 0;
}

/************************************************************
    Hard2Player()

Called when the hard setting is chosen in the 2 player mode
************************************************************/
int CVICALLBACK Hard2Player (int panel, int control, int event,
        void *callbackData, int eventData1, int eventData2)
{
    switch (event)
        {
        case EVENT_COMMIT:
            timerwanted = 1;
            if ((panel1 = LoadPanel (0, "Connect 4.uir", PANEL)) < 0)
                return -1;
            SetPanelAttribute (panel1, ATTR_TITLE, "Connect 4");
            SetCtrlAttribute (panel1, PANEL_TIMER, ATTR_VISIBLE, 1);
            ResetTimer (panel1, PANEL_TIMER);
            DisplayPanel (panel1);
            SetActivePanel (panel1);
            DiscardPanel(panel5);

            break;
        }
    return 0;
}

/************************************************************
    Easy2Player()

Called when the easy setting is chosen in the 2 player mode
************************************************************/
int CVICALLBACK Easy2Player (int panel, int control, int event,
        void *callbackData, int eventData1, int eventData2)
{
    switch (event)
        {
        case EVENT_COMMIT:
            timerwanted = 0;
            if ((panel1 = LoadPanel (0, "Connect 4.uir", PANEL)) < 0)
                return -1;
            SetPanelAttribute (panel1, ATTR_TITLE, "Connect 4");
            SetCtrlAttribute (panel1, PANEL_TEXTMSG_2, ATTR_VISIBLE, 0);
            SetCtrlAttribute (panel1, PANEL_TEXTMSG_3, ATTR_VISIBLE, 0);
            SetCtrlAttribute (panel1, PANEL_TEXTMSG_4, ATTR_VISIBLE, 0);
            DisplayPanel (panel1);
            SetActivePanel (panel1);
            DiscardPanel(panel5);

            break;
        }
    return 0;
}


/************************************************************
    Column1()

Called when player selects the first columm and depending
on what row is currently available, it stores the selection
in the board
************************************************************/
int CVICALLBACK Column1 (int panel, int control, int event,
        void *callbackData, int eventData1, int eventData2)
{
    int currentcolor;
    int x;
    if (player == 1)
    {
        currentcolor =  player1color;
    }
    else if (player == 2)
    {
        currentcolor =  player2color;
    }
    else
    {
        currentcolor =  computerplayercolor;
    }
    columnpick = 1;
    switch (event)
        {
        case EVENT_COMMIT:
            if (rowindicator[columnpick] == 1) //row 1 is next row available
            {
                SetCtrlAttribute (panel1, PANEL_LED_36, ATTR_OFF_COLOR, currentcolor);
                x = loadboard(columnpick,1);
            }
            else if (rowindicator[columnpick] == 2)
            {
                SetCtrlAttribute (panel1, PANEL_LED_29, ATTR_OFF_COLOR, currentcolor);
                x = loadboard(columnpick,2);

            }
            else if (rowindicator[columnpick] == 3)
            {
                SetCtrlAttribute (panel1, PANEL_LED_22, ATTR_OFF_COLOR, currentcolor);
                x = loadboard(columnpick,3);

            }
            else if (rowindicator[columnpick] == 4)
            {
                SetCtrlAttribute (panel1, PANEL_LED_15, ATTR_OFF_COLOR, currentcolor);
                x = loadboard(columnpick,4);

            }
            else if (rowindicator[columnpick] == 5)
            {
                SetCtrlAttribute (panel1, PANEL_LED_8, ATTR_OFF_COLOR, currentcolor);
                x = loadboard(columnpick,5);

            }
            else if (rowindicator[columnpick] == 6)
            {
                SetCtrlAttribute (panel1, PANEL_LED_1, ATTR_OFF_COLOR, currentcolor);
                x = loadboard(columnpick,6);

            }
            else
            {
                //column is full
                x = 0;
            }
            if (x == 0)
            {
                if (rowindicator[columnpick] <= row)
                {
                    rowindicator[columnpick]++;
                }
                else
                {
                    //column is full
                }
                changeplayer();
            }
            DefaultCtrl  (panel1, PANEL_PICTUREBUTTON_1);

            break;
        }
    return 0;
}


/************************************************************
    Column2()

Called when player selects the second columm and depending
on what row is currently available, it stores the selection
in the board
************************************************************/
int CVICALLBACK Column2 (int panel, int control, int event,
        void *callbackData, int eventData1, int eventData2)
{
    int currentcolor;
    int x;
    if (player == 1)
    {
        currentcolor =  player1color;
    }
    else if (player == 2)
    {
        currentcolor =  player2color;
    }
    else
    {
        currentcolor =  computerplayercolor;
    }
    columnpick = 2;
    switch (event)
        {
        case EVENT_COMMIT:
            if (rowindicator[columnpick] == 1) //row 1 is next row available
            {
                SetCtrlAttribute (panel1, PANEL_LED_37, ATTR_OFF_COLOR, currentcolor);
                x = loadboard(columnpick,1);
            }
            else if (rowindicator[columnpick] == 2)
            {
                SetCtrlAttribute (panel1, PANEL_LED_30, ATTR_OFF_COLOR, currentcolor);
                x = loadboard(columnpick,2);

            }
            else if (rowindicator[columnpick] == 3)
            {
                SetCtrlAttribute (panel1, PANEL_LED_23, ATTR_OFF_COLOR, currentcolor);
                x = loadboard(columnpick,3);

            }
            else if (rowindicator[columnpick] == 4)
            {
                SetCtrlAttribute (panel1, PANEL_LED_16, ATTR_OFF_COLOR, currentcolor);
                x = loadboard(columnpick,4);

            }
            else if (rowindicator[columnpick] == 5)
            {
                SetCtrlAttribute (panel1, PANEL_LED_9, ATTR_OFF_COLOR, currentcolor);
                x = loadboard(columnpick,5);

            }
            else if (rowindicator[columnpick] == 6)
            {
                SetCtrlAttribute (panel1, PANEL_LED_2, ATTR_OFF_COLOR, currentcolor);
                x = loadboard(columnpick,6);

            }
            else
            {
                //column is full
                x = 0;
            }
            if (x == 0)
            {
                if (rowindicator[columnpick] <= row)
                {
                    rowindicator[columnpick]++;
                }
                else
                {
                    //column is full
                }
                changeplayer();
            }
            DefaultCtrl  (panel1, PANEL_PICTUREBUTTON_2);

            break;
        }
    return 0;
}

/************************************************************
    Column3()

Called when player selects the third columm and depending
on what row is currently available, it stores the selection
in the board
************************************************************/
int CVICALLBACK Column3 (int panel, int control, int event,
        void *callbackData, int eventData1, int eventData2)
{
    int currentcolor;
    int x;
    if (player == 1)
    {
        currentcolor =  player1color;
    }
    else if (player == 2)
    {
        currentcolor =  player2color;
    }
    else
    {
        currentcolor =  computerplayercolor;
    }
    columnpick = 3;
    switch (event)
        {
```

```
        case EVENT_COMMIT:
            if (rowindicator[columnpick] == 1) //row 1 is next row available
            {
                SetCtrlAttribute (panel1, PANEL_LED_38, ATTR_OFF_COLOR, currentcolor);
                x = loadboard(columnpick,1);
            }
            else if (rowindicator[columnpick] == 2)
            {
                SetCtrlAttribute (panel1, PANEL_LED_31, ATTR_OFF_COLOR, currentcolor);
                x = loadboard(columnpick,2);

            }
            else if (rowindicator[columnpick] == 3)
            {
                SetCtrlAttribute (panel1, PANEL_LED_24, ATTR_OFF_COLOR, currentcolor);
                x = loadboard(columnpick,3);

            }
            else if (rowindicator[columnpick] == 4)
            {
                SetCtrlAttribute (panel1, PANEL_LED_17, ATTR_OFF_COLOR, currentcolor);
                x = loadboard(columnpick,4);

            }
            else if (rowindicator[columnpick] == 5)
            {
                SetCtrlAttribute (panel1, PANEL_LED_10, ATTR_OFF_COLOR, currentcolor);
                x = loadboard(columnpick,5);

            }
            else if (rowindicator[columnpick] == 6)
            {
                SetCtrlAttribute (panel1, PANEL_LED_3, ATTR_OFF_COLOR, currentcolor);
                x = loadboard(columnpick,6);

            }
            else
            {
                //column is full
                x = 0;
            }
            if (x == 0)
            {
                if (rowindicator[columnpick] <= row)
                {
                    rowindicator[columnpick]++;
                }
                else
                {
                    //column is full
                }
                changeplayer();
            }
            DefaultCtrl  (panel1, PANEL_PICTUREBUTTON_3);

            break;
        }
    return 0;
}

/************************************************************
    Column4()

Called when player selects the fourth columm and depending
on what row is currently available, it stores the selection
in the board
************************************************************/
int CVICALLBACK Column4 (int panel, int control, int event,
        void *callbackData, int eventData1, int eventData2)
{
    int currentcolor;
    int x;
    if (player == 1)
    {
        currentcolor =  player1color;
    }
    else if (player == 2)
    {
        currentcolor =  player2color;
    }
    else
    {
        currentcolor =  computerplayercolor;
    }
    columnpick = 4;
    switch (event)
        {
        case EVENT_COMMIT:
            if (rowindicator[columnpick] == 1) //row 1 is next row available
            {
                SetCtrlAttribute (panel1, PANEL_LED_39, ATTR_OFF_COLOR, currentcolor);
                x = loadboard(columnpick,1);
            }
            else if (rowindicator[columnpick] == 2)
            {
                SetCtrlAttribute (panel1, PANEL_LED_32, ATTR_OFF_COLOR, currentcolor);
                x = loadboard(columnpick,2);

            }
            else if (rowindicator[columnpick] == 3)
            {
                SetCtrlAttribute (panel1, PANEL_LED_25, ATTR_OFF_COLOR, currentcolor);
                x = loadboard(columnpick,3);

            }
            else if (rowindicator[columnpick] == 4)
            {
                SetCtrlAttribute (panel1, PANEL_LED_18, ATTR_OFF_COLOR, currentcolor);
                x = loadboard(columnpick,4);

            }
            else if (rowindicator[columnpick] == 5)
            {
                SetCtrlAttribute (panel1, PANEL_LED_11, ATTR_OFF_COLOR, currentcolor);
                x = loadboard(columnpick,5);

            }
            else if (rowindicator[columnpick] == 6)
            {
                SetCtrlAttribute (panel1, PANEL_LED_4, ATTR_OFF_COLOR, currentcolor);
                x = loadboard(columnpick,6);

            }
            else
            {
                //column is full
                x = 0;
            }
            if (x == 0)
            {
                if (rowindicator[columnpick] <= row)
                {
                    rowindicator[columnpick]++;
                }
                else
                {
                    //column is full
                }
                changeplayer();
            }
            DefaultCtrl  (panel1, PANEL_PICTUREBUTTON_4);

            break;
        }
    return 0;
}

/************************************************************
    Column5()

Called when player selects the fifth columm and depending
on what row is currently available, it stores the selection
in the board
************************************************************/
int CVICALLBACK Column5 (int panel, int control, int event,
        void *callbackData, int eventData1, int eventData2)
{
    int currentcolor;
    int x;
    if (player == 1)
    {
        currentcolor =  player1color;
    }
    else if (player == 2)
    {
        currentcolor =  player2color;
    }
    else
    {
        currentcolor =  computerplayercolor;
    }
    columnpick = 5;
    switch (event)
        {
        case EVENT_COMMIT:
            if (rowindicator[columnpick] == 1) //row 1 is next row available
            {
                SetCtrlAttribute (panel1, PANEL_LED_40, ATTR_OFF_COLOR, currentcolor);
                x = loadboard(columnpick,1);
            }
            else if (rowindicator[columnpick] == 2)
            {
                SetCtrlAttribute (panel1, PANEL_LED_33, ATTR_OFF_COLOR, currentcolor);
                x = loadboard(columnpick,2);

            }
            else if (rowindicator[columnpick] == 3)
            {
                SetCtrlAttribute (panel1, PANEL_LED_26, ATTR_OFF_COLOR, currentcolor);
                x = loadboard(columnpick,3);

            }
            else if (rowindicator[columnpick] == 4)
            {
                SetCtrlAttribute (panel1, PANEL_LED_19, ATTR_OFF_COLOR, currentcolor);
                x = loadboard(columnpick,4);

            }
            else if (rowindicator[columnpick] == 5)
            {
                SetCtrlAttribute (panel1, PANEL_LED_12, ATTR_OFF_COLOR, currentcolor);
                x = loadboard(columnpick,5);

            }
            else if (rowindicator[columnpick] == 6)
            {
                SetCtrlAttribute (panel1, PANEL_LED_5, ATTR_OFF_COLOR, currentcolor);
                x = loadboard(columnpick,6);

            }
            else
            {
                //column is full
                x = 0;
            }
            if (x == 0)
            {
                if (rowindicator[columnpick] <= row)
                {
                    rowindicator[columnpick]++;
                }
                else
                {
                    //column is full
                }
                changeplayer();
            }
            DefaultCtrl  (panel1, PANEL_PICTUREBUTTON_5);

            break;
        }
    return 0;
}

/************************************************************
    Column6()

Called when player selects the sixth columm and depending
on what row is currently available, it stores the selection
in the board
************************************************************/
int CVICALLBACK Column6 (int panel, int control, int event,
        void *callbackData, int eventData1, int eventData2)
{
    int currentcolor;
    int x;
    if (player == 1)
    {
        currentcolor =  player1color;
    }
    else if (player == 2)
    {
        currentcolor =  player2color;
    }
    else
    {
        currentcolor =  computerplayercolor;
    }
    columnpick = 6;
    switch (event)
        {
        case EVENT_COMMIT:
            if (rowindicator[columnpick] == 1) //row 1 is next row available
            {
                SetCtrlAttribute (panel1, PANEL_LED_41, ATTR_OFF_COLOR, currentcolor);
                x = loadboard(columnpick,1);
            }
            else if (rowindicator[columnpick] == 2)
            {
                SetCtrlAttribute (panel1, PANEL_LED_34, ATTR_OFF_COLOR, currentcolor);
                x = loadboard(columnpick,2);

            }
            else if (rowindicator[columnpick] == 3)
            {
                SetCtrlAttribute (panel1, PANEL_LED_27, ATTR_OFF_COLOR, currentcolor);
                x = loadboard(columnpick,3);

            }
            else if (rowindicator[columnpick] == 4)
            {
                SetCtrlAttribute (panel1, PANEL_LED_20, ATTR_OFF_COLOR, currentcolor);
                x = loadboard(columnpick,4);

            }
            else if (rowindicator[columnpick] == 5)
            {
                SetCtrlAttribute (panel1, PANEL_LED_13, ATTR_OFF_COLOR, currentcolor);
                x = loadboard(columnpick,5);

            }
            else if (rowindicator[columnpick] == 6)
            {
                SetCtrlAttribute (panel1, PANEL_LED_6, ATTR_OFF_COLOR, currentcolor);
                x = loadboard(columnpick,6);

            }
            else
            {
                //column is full
                x = 0;
            }
            if (x == 0)
            {
                if (rowindicator[columnpick] <= row)
                {
                    rowindicator[columnpick]++;
                }
                else
                {
                    //column is full
                }
                changeplayer();
            }
            DefaultCtrl  (panel1, PANEL_PICTUREBUTTON_6);

            break;
        }
    return 0;
}

/**********************************************************
    Column7()

Called when player selects the seventh columm and depending
on what row is currently available, it stores the selection
in the board
**********************************************************/
int CVICALLBACK Column7 (int panel, int control, int event,
        void *callbackData, int eventData1, int eventData2)
{
    int currentcolor;
    int x;
    if (player == 1)
    {
        currentcolor =  player1color;
    }
    else if (player == 2)
    {
        currentcolor =  player2color;
    }
    else
    {
        currentcolor =  computerplayercolor;
    }
    columnpick = 7;
    switch (event)
        {
        case EVENT_COMMIT:
            if (rowindicator[columnpick] == 1) //row 1 is next row available
            {
                SetCtrlAttribute (panel1, PANEL_LED_42, ATTR_OFF_COLOR, currentcolor);
                x = loadboard(columnpick,1);
            }
            else if (rowindicator[columnpick] == 2)
            {
                SetCtrlAttribute (panel1, PANEL_LED_35, ATTR_OFF_COLOR, currentcolor);
                x = loadboard(columnpick,2);

            }
            else if (rowindicator[columnpick] == 3)
            {
                SetCtrlAttribute (panel1, PANEL_LED_28, ATTR_OFF_COLOR, currentcolor);
                x = loadboard(columnpick,3);

            }
            else if (rowindicator[columnpick] == 4)
            {
                SetCtrlAttribute (panel1, PANEL_LED_21, ATTR_OFF_COLOR, currentcolor);
                x = loadboard(columnpick,4);

            }
            else if (rowindicator[columnpick] == 5)
            {
                SetCtrlAttribute (panel1, PANEL_LED_14, ATTR_OFF_COLOR, currentcolor);
                x = loadboard(columnpick,5);

            }
            else if (rowindicator[columnpick] == 6)
            {
                SetCtrlAttribute (panel1, PANEL_LED_7, ATTR_OFF_COLOR, currentcolor);
                x = loadboard(columnpick,6);

            }
            else
            {
                //column is full
                x = 0;
            }
            if (x == 0)
            {
                if (rowindicator[columnpick] <= row)
                {
                    rowindicator[columnpick]++;
                }
                else
                {
                    //column is full
                }
                changeplayer();
            }
            DefaultCtrl  (panel1, PANEL_PICTUREBUTTON_7);

            break;
        }
    return 0;
}
```