Republic of the Philippines
BOHOL ISLAND STATE UNIVERSITY
MAIN CAMPUS
Tel: 038-4113289 Telfax: 038-5017516
6300 Tagbilaran City

Vision: A premier S&T university for the formation of world class and virtuous human resource for sustainable development in Bohol and the Country.

Mission: BISU is committed to provide quality higher education in the arts and sciences, as well as in the professional and technological fields; undertake research and development, and extension services for the sustainable development of Bohol and the country.

Project #2

CpE 515o – Object Oriented Programming

Connect 4
(Inheritance)

Submitted by:

Mejorada, John Kenneth

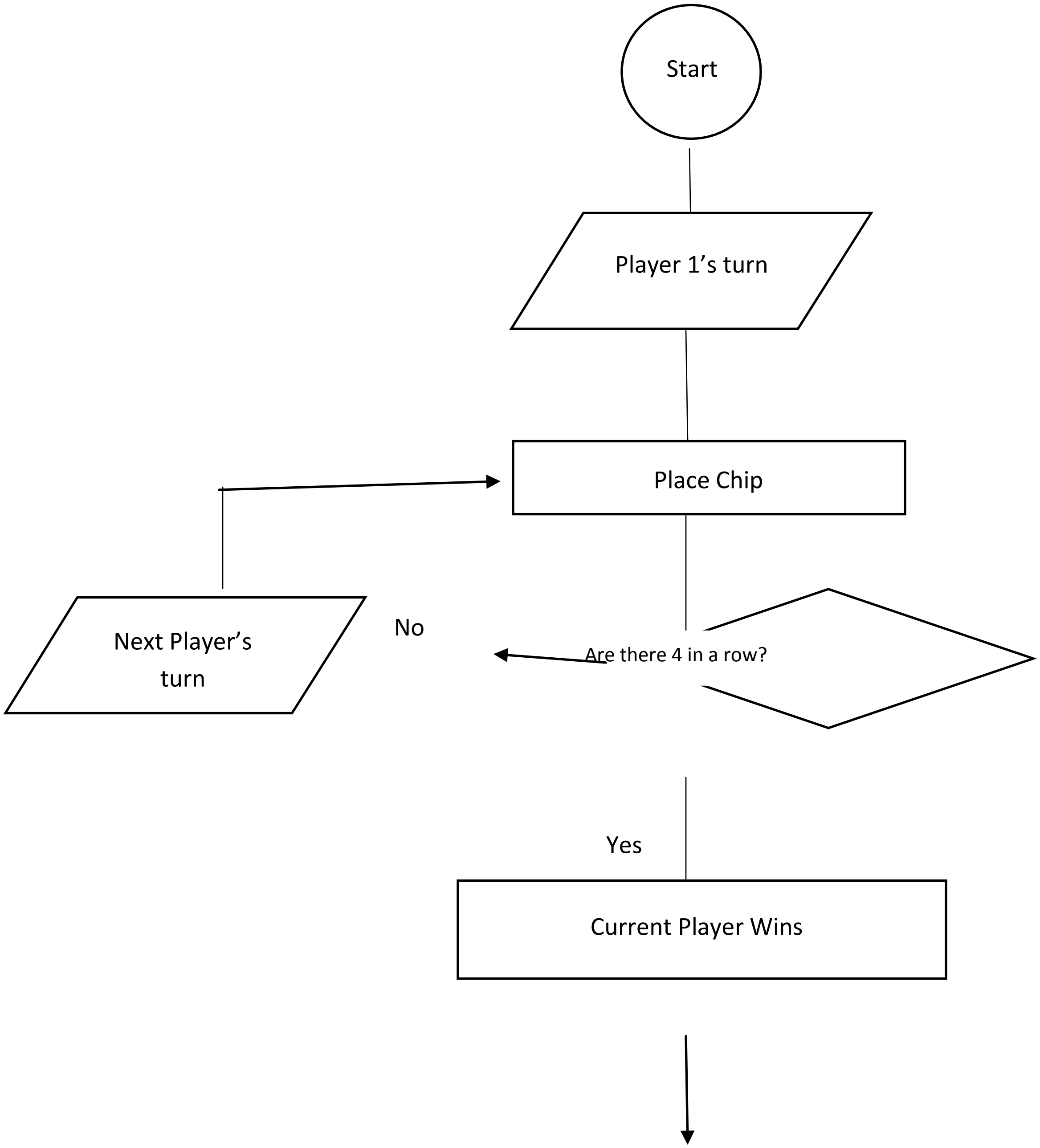
Submitted to:

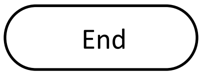
Engr. Edgar Uy II

I. Objectives and Goals

Objective of this project is to create a derived class and do the concept of inheritance from an existing base class. The base class is from my Minimax Algorithm. The purpose of this application is to let the players play whenever they want wherever they like without the game board.

II. Flowchart





III. Algorithm

- Input data
- Determine minimum and maximum
- For each data point:
  - Find minimum score of player
  - Find maximum score of player
- For each cluster j=1….K:
  - New centroid Cj = mean of all points Xi assigned to cluster j in previous step
- Stop when none of the cluster assignments change

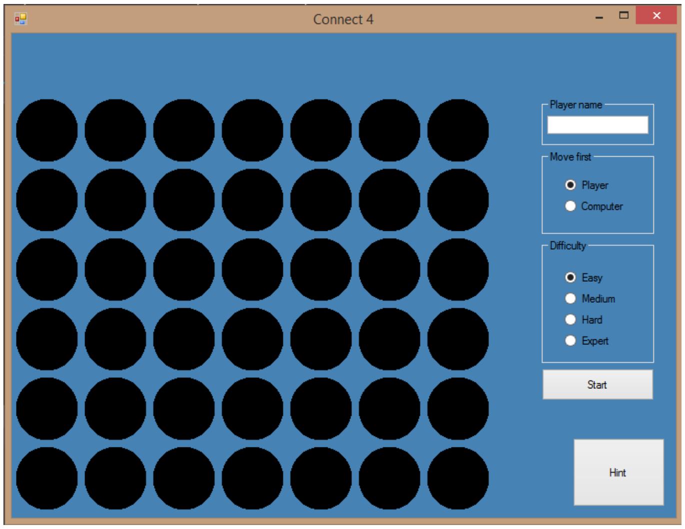
IV. Algorithm Analysis

This algorithm will calculate the distance between the data points and the cluster and will group the data based on the minimum distance.

1. Begin
2. Input your data points (score<space>name)
3. Determine the initial value of cluster (usually the lowest and highest value)
4. Calculate the distance between the data and the cluster using the Euclidean method (distance formula)

   $$Distance\ [(x,y),(a,b)] = \sqrt{(x-a)^2 + (y-b)^2}$$
5. Assign the data points to its nearest cluster
6. Update cluster centroid values
7. Repeat 4,5,6 until no changes between the cluster-data assignment.

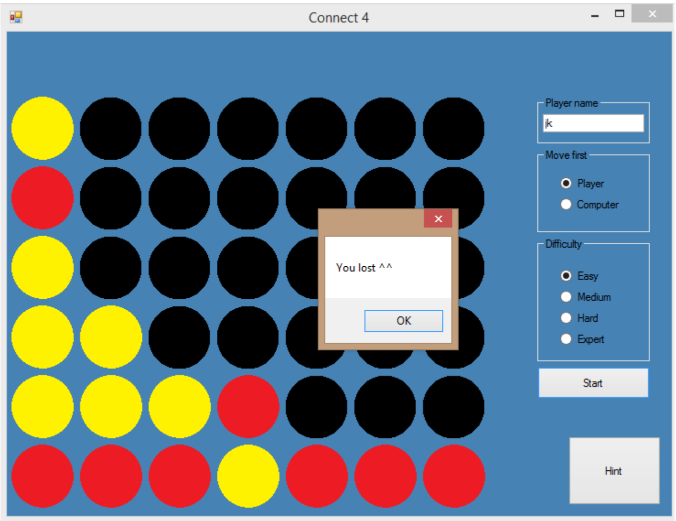
V. Snapshots of the Functionality



Player name

Move first
● Player
○ Computer

Difficulty
● Easy
○ Medium
○ Hard
○ Expert

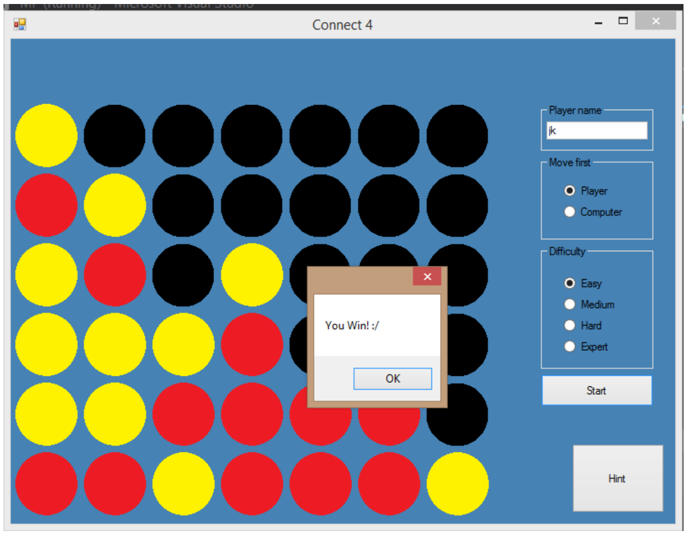


Player name: jk

Move first
● Player
○ Computer

Difficulty
● Easy
○ Medium
○ Hard
○ Expert

You lost ^^



Player name: jk

Move first
● Player
○ Computer

Difficulty
● Easy
○ Medium
○ Hard
○ Expert

You Win! :/

VI. Code

// John Kenneth MejoradaBSCpE 5
// Application of Minimax algorithm for Connect 4

BASE CLASS

```
using System;
usingSystem.Collections.Generic;
usingSystem.ComponentModel;
usingSystem.Data;
usingSystem.Drawing;
usingSystem.Linq;
usingSystem.Text;
usingSystem.Threading.Tasks;
usingSystem.Windows.Forms;
using System.IO;

namespace MP
{
classMiniMax
    {
publicvoidSetDecision(intDep, bool Maxim)
        {
            Depth = Dep;
    Maximize = Maxim;
        }
publicvoidSetHeuristicValue(int Player, int[,] State)
        {

inttmp = 0, Coin = Player;
for (int x = 1; x <= 6; x++)
            {
for (int y = 1; y <= 7; y++)
if (State[x, y] == Coin)
                    {
for (inti = x; i<= 6; i++)
                        {
if (State[i, y] == Coin)
                                ++tmp;
else
break;
                        }
for (inti = x - 1; i>= 1; i--)
                        {
if (State[i, y] == Coin)
                                ++tmp;
else
break;
                        }
                        ++cnt[tmp];
tmp = 0;
for (inti = y; i<= 7; i++)
                        {
if (State[x, i] == Coin)
                                ++tmp;
else
break;
                        }
for (inti = y - 1; i>= 1; i--)
                        {
if (State[x, i] == Coin)
                                ++tmp;
else
break;
                        }
                        ++cnt[tmp];
tmp = 0;
for (int a = x, b = y; a <= 6 && b <= 7; ++a, ++b)
                        {
if (State[a, b] == Coin)
                                ++tmp;
else
break;
                        }
for (int a = x - 1, b = y - 1; a >= 1 && b >= 1; --a, --b)
                        {
if (State[a, b] == Coin)
                                ++tmp;
else
break;
                        }
                        ++cnt[tmp];
tmp = 0;
for (int a = x, b = y; a <= 6 && b >= 1; ++a, --b)
                        {
if (State[a, b] == Coin)
                                ++tmp;
else
break;
                        }
for (int a = x - 1, b = y + 1; a >= 1 && b <= 7; --a, ++b)
                        {
if (State[a, b] == Coin)
                                ++tmp;
else
break;
                        }
                        ++cnt[tmp];
tmp = 0;
                    }
            }

for (inti = 1; i<= 4; i++)
            {
cnt[i] = Math.Min(cnt[i], 1);
            }

        }
protectedint[] cnt = newint[15];
protectedint Depth;
protectedbool Maximize;
    }

}
```

DERIVED CLASS

```
using System;
usingSystem.Collections.Generic;
usingSystem.ComponentModel;
usingSystem.Data;
usingSystem.Drawing;
usingSystem.Linq;
usingSystem.Text;
usingSystem.Threading.Tasks;
usingSystem.Windows.Forms;
using System.IO;

namespace MP
{
classPlayGame : MiniMax
    {
publicintOptimalColumn;
MakeMove Move = newMakeMove();
public Board GameBoard = new Board();
publicintMaxDepth;

publicintMakeDecision()
        {
if (GameBoard.CheckWinner(1))
return (int)1e8 * Depth;
if (GameBoard.CheckWinner(-1))
return -(int)1e8 * Depth;
if (Depth == 0)
returnHeuristicValue((Maximize ? 1 : -1), GameBoard.State);
if (Maximize)
            {
int Score = int.MinValue;
for (inti = 1; i<= 7; i++)
                {
int Row = Move.IsPossible(i, GameBoard.State);
if (Row != -1)
                    {
GameBoard.State[Row, i] = 1;

PlayGame Play = newPlayGame();
Play.SetDecision(Depth - 1, !Maximize);
int Value = Play.MakeDecision();

GameBoard.State[Row, i] = 0;
if (Score < Value)
                        {
                            Score = Value;
if (Depth == MaxDepth)
OptimalColumn = i;
                        }
                    }
                }
return Score;
            }
else
            {
int Score = int.MaxValue;
for (inti = 1; i<= 7; i++)
                {
int Row = Move.IsPossible(i, GameBoard.State);
if (Row != -1)
                    {
GameBoard.State[Row, i] = -1;

PlayGame Play = newPlayGame();
Play.SetDecision(Depth - 1, !Maximize);
int Value = Play.MakeDecision();


GameBoard.State[Row, i] = 0;
                        Score = Math.Min(Score, Value);
                    }
                }
return Score;
            }
        }

publicintHeuristicValue(int Player, int[,] State)
        {
returncnt[1] * 10 + cnt[2] * 1000 + cnt[3] * 100000 + cnt[4] * 10000000;
        }
    }
}
```

MAKEMOVE

```
using System;
usingSystem.Collections.Generic;
usingSystem.Linq;
usingSystem.Text;
usingSystem.Threading.Tasks;

namespace MP
{
classMakeMove
    {
publicintPlayerMove(int x, List<Tuple<int, int>>PositionsOnX)
        {
for (inti = 0; i< 7; i++) if (x >= PositionsOnX[i].Item1 && x <= PositionsOnX[i].Item2)
returni + 1;
return -1;
        }

publicintPCMove(int Depth, int[,] GameBoard)
        {
PlayGame Game = newPlayGame();
Game.GameBoard.State = GameBoard;
Game.MaxDepth = Depth;
Game.SetDecision(Depth, true);
Game.MakeDecision();
returnGame.OptimalColumn;
        }

publicintIsPossible(int Column, int[,] GameBoard)
        {
for (inti = 6; i> 0; i--)
if (GameBoard[i, Column] == 0)
returni;
return -1;
        }
    }
}
```

BOARD

```
using System;
usingSystem.Collections.Generic;
usingSystem.ComponentModel;
usingSystem.Data;
usingSystem.Drawing;
usingSystem.Linq;
usingSystem.Text;
usingSystem.Threading.Tasks;
usingSystem.Windows.Forms;
using System.IO;

namespace MP
{
    classBoard
    {
        publicint[,] State = newint[7, 8];
        PictureBox Cell;
        publicvoidColorCell(Form1 obj, int Row, int Column, bool Player)
        {
            intstpx = 68;
            intstpy = 69;
            Cell = newPictureBox();
            Cell.Size = newSize(68, 69);
            stringCellPath;
            if (!Player)
                CellPath = Path.Combine(Environment.CurrentDirectory,
@"imgs\RED.jpg");
            else
                CellPath = Path.Combine(Environment.CurrentDirectory,
@"imgs\YELLOW.jpg");
            Cell.ImageLocation = CellPath;
            Cell.Location = newPoint((Column - 1) * (stpx + 1 + 6) + 1 + 4, (Row
- 1) * (stpy + 1 + 6) + 1 + 70);
            obj.Controls.Add(Cell);
        }

        publicboolCheckWinner(int Player)
        {
            int ret = 0, tmp = 0, Coin = Player;

            for (int x = 1; x <= 6; x++)
            {
                for (int y = 1; y <= 7; y++)
                    if (State[x, y] == Coin)
                    {
                        for (inti = x; i<= 6; i++)
                            if (State[i, y] == Coin)
                                ++tmp;
                            else
                                break;
                        for (inti = x - 1; i>= 1; i--)
                            if (State[i, y] == Coin)
                                ++tmp;
                            else
                                break;
                        ret = Math.Max(ret, tmp);
                        tmp = 0;
                        for (inti = y; i<= 7; i++)
```

```
                            if (State[x, i] == Coin)
                                ++tmp;
                            else
                                break;
                        for (inti = y - 1; i>= 1; i--)
                            if (State[x, i] == Coin)
                                ++tmp;
                            else
                                break;
                        ret = Math.Max(ret, tmp);
                        tmp = 0;
                        for (int a = x, b = y; a <= 6 && b <= 7; ++a,
++b)

                            if (State[a, b] == Coin)
                                ++tmp;
                            else
                                break;
                        for (int a = x - 1, b = y - 1; a >= 1 && b >= 1;
--a, --b)

                            if (State[a, b] == Coin)
                                ++tmp;
                            else
                                break;
                        ret = Math.Max(ret, tmp);
                        tmp = 0;
                        for (int a = x, b = y; a <= 6 && b >= 1; ++a, --
b)

                            if (State[a, b] == Coin)
                                ++tmp;
                            else
                                break;
                        for (int a = x - 1, b = y + 1; a >= 1 && b <= 7;
--a, ++b)

                            if (State[a, b] == Coin)
                                ++tmp;
                            else
                                break;
                        ret = Math.Max(ret, tmp);
                        tmp = 0;
                        if (ret > 3)
                            returntrue;
                    }
            }
            returnfalse;
        }

    }
}
```