using System;

using System.Drawing;

using System.Windows.Forms;

namespace Test\_3

{

public partial class Form1 : Form

{

//Global Declarations:

Image MySoldierImg;

Image EnemySoldierImg;

Image SecEnemySoldierImg;

//Healthpoints

int MySoldiersHP = 10;

int MyEnemy1HP = 10;

int MyEnemy2HP = 10;

Random randomNumberGenerator = new Random();

int rowNum, colNum, EnemyRowNum, EnemyColNum, SecEnemyColNum, SecEnemyRowNum,

TargetColNum, TargetRowNum, DistanceToEnemy1, DistanceToEnemy2, DistanceEnemy1ToEnemy2,

MySoldiersDirection, MyEnemySoldiersDirection, MySecEnemySoldiersDirection,

Enemy1TargetColNum, Enemy1TargetRowNum, Enemy2TargetColNum, Enemy2TargetRowNum;

int AttackRange = 2;

int moveRight = 1;

int moveLeft = 2;

int moveUp = 3;

int moveDown = 4;

// create my instance of the transparent grid

MyTransparentDataGrid MyGrid = new MyTransparentDataGrid();

//Corner Escape Variables:

int SoldierCornerEscapeCounter, SoldierCornerEscape1stDirection,

Enemy1CornerEscapeCounter, Enemy1CornerEscape1stDirection,

Enemy2CornerEscapeCounter, Enemy2CornerEscape1stDirection;

// a string to store the corner someone is getting stuck in

string SoldierCornerLabel, Enemy1CornerLabel, Enemy2CornerLabel;

public Form1()

{// initialise the form

InitializeComponent();

//Soldier Image:

MySoldierImg = Properties.Resources.Ranged\_Soldier;

//First Enemy Soldier Image:

EnemySoldierImg = Properties.Resources.General;

//Second Enemy Image:

SecEnemySoldierImg = Properties.Resources.PostmanPat;

//My Soldiers Position:

rowNum = randomNumberGenerator.Next(0, 19);

colNum = randomNumberGenerator.Next(0, 19);

//My Enemy's Position:

EnemyColNum = randomNumberGenerator.Next(0, 19);

EnemyRowNum = randomNumberGenerator.Next(0, 19);

// ensure the enemy is not on the same spot as soldier

while ((colNum == EnemyColNum) && (rowNum == EnemyRowNum))

{

EnemyColNum = randomNumberGenerator.Next(0, 19);

EnemyRowNum = randomNumberGenerator.Next(0, 19);

}

//My Second Enemy's Position:

SecEnemyColNum = randomNumberGenerator.Next(0, 19);

SecEnemyRowNum = randomNumberGenerator.Next(0, 19);

// ensure the 2nd enemy is not on the same spot as the first

while((SecEnemyColNum == EnemyColNum) && (SecEnemyRowNum == EnemyRowNum))

{

SecEnemyColNum = randomNumberGenerator.Next(0, 19);

SecEnemyRowNum = randomNumberGenerator.Next(0, 19);

}

}// end of Form initialise

// create my customized grid in memory, that is transparent, with a picture as a background image

public class MyTransparentDataGrid : DataGridView

{

private Image MyPicture; // this will be the picture of the aerial map

// constructor (initialisor) for MyTransparentDataGrid.... sets MyPicture

public MyTransparentDataGrid()

{

//this.MyPicture = Image.FromFile("C:\\Users\\Jhieb\\Desktop\\Project test\\Map.JPG");

this.MyPicture = Properties.Resources.Map;

}

// accessor for property MyBackgroundImage.... use MyPicture

public Image MyBackgroundImage // create a public property on the transparent grid... which uses MyPicture

{

get { return MyPicture; }

set { MyPicture = value; }

}

// override the PaintBackGround method of the grid... as follows:

protected override void PaintBackground(System.Drawing.Graphics graphics,

System.Drawing.Rectangle clipBounds,

System.Drawing.Rectangle gridBounds)

{

base.PaintBackground(graphics, clipBounds, gridBounds); // generally, use the PaintBackground method as is...

//graphics.FillRectangle(Brushes.Black, gridBounds); //...but make a black rectangle over the grid area (gridbounds)... not needed

graphics.DrawImage(this.MyBackgroundImage, gridBounds); //...and then draw MyBackgroundImage (property created above) over the same grid area

// ... and also, make the entire grid transparent

foreach (DataGridViewColumn col in this.Columns) // loop thru the columns, making each transaparent

{ col.DefaultCellStyle.BackColor = Color.Transparent; }

this.EnableHeadersVisualStyles = false; // hide the grid's headings

this.ColumnHeadersDefaultCellStyle.BackColor = Color.Transparent; // make column headings transparent

this.RowHeadersDefaultCellStyle.BackColor = Color.Transparent; // make row headings transparent

}

} //end of my custom grid created in memory

private void BtnExit\_Click(object sender, EventArgs e)

{

Close();

}

public void DisplayMyGrid()

{

// create my instance of the transparent grid

//MyTransparentDataGrid MyGrid = new MyTransparentDataGrid();

// add my grid to the form

this.Controls.Add(MyGrid);

// adding 20 image columns to an empty grid

for (int i = 0; i < 20; i++)

{ // create a new image column - set to null, let images be stretched to fill the cell

DataGridViewImageColumn imageColumn = new DataGridViewImageColumn();

imageColumn.DefaultCellStyle.NullValue = null;

imageColumn.ImageLayout = DataGridViewImageCellLayout.Stretch;

MyGrid.Columns.Add(imageColumn);

}

//Add 20 rows of the above coloumns:

for (int i = 0; i < 20; i++)

{ MyGrid.Rows.Add(); }

//set column widths

for (int i = 0; i < 20; i++)

{ MyGrid.Columns[i].Width = 25; }

//set row heights

for (int i = 0; i < 20; i++)

{ MyGrid.Rows[i].Height = 25; }

//set location & size of grid

MyGrid.Location = new System.Drawing.Point(20, 20); // top left corner position

MyGrid.Size = new System.Drawing.Size(507, 507); // width, height

//Making the grid lines transparent.

MyGrid.CellBorderStyle = DataGridViewCellBorderStyle.None;

//Clear the users selection/cursor ....which starts in the top left cell

MyGrid.RowsDefaultCellStyle.SelectionBackColor = System.Drawing.Color.Transparent;

// make the row-headers and column-headers very small (to 'hide' it)

MyGrid.ColumnHeadersHeight = 4;

MyGrid.RowHeadersWidth = 4;

// dont want scroll-bars on the grid

MyGrid.ScrollBars = ScrollBars.None;

}

private void BtnPlay\_Click(object sender, EventArgs e)

{

GameTimer.Start();

}

private void BtnPause\_Click(object sender, EventArgs e)

{

GameTimer.Stop();

}

private void Form1\_Load(object sender, EventArgs e)

{

DisplayMyGrid();

}

// ----------------------------------------------------------------------------

private void GameTimer\_Tick(object sender, EventArgs e)

{

//Stop the GAME if 2 is dead

if (MyEnemy1HP == 0 && MyEnemy2HP == 0) { GameTimer.Stop(); }

if (MySoldiersHP == 0 && MyEnemy1HP == 0) { GameTimer.Stop(); }

if (MySoldiersHP == 0 && MyEnemy2HP == 0) { GameTimer.Stop(); }

//Clear the cell that my soldier is going to

MyGrid.Rows[rowNum].Cells[colNum].Value = null;

//Clear the cell that enemy1 is going to

MyGrid.Rows[EnemyRowNum].Cells[EnemyColNum].Value = null;

//Clear the cell that my second enemy is going to

MyGrid.Rows[SecEnemyRowNum].Cells[SecEnemyColNum].Value = null;

//find distance between soldier and enemy1...if they're both alive

DistanceToEnemy1 = 100; //default... assuming one or both is dead

if ((MyEnemy1HP > 0 ) && (MySoldiersHP > 0 )) { DistanceToEnemy1 = Math.Abs(colNum - EnemyColNum) + Math.Abs(rowNum - EnemyRowNum); }

//find distance between soldier and enemy2...if they're both alive

DistanceToEnemy2 = 100; //default... assuming one or both is dead

if ((MyEnemy2HP > 0) && (MySoldiersHP > 0)) { DistanceToEnemy2 = Math.Abs(colNum - SecEnemyColNum) + Math.Abs(rowNum - SecEnemyRowNum); }

//find distance between enemy1 and enemy2...if they're both alive

DistanceEnemy1ToEnemy2 = 100; //default... assuming one or both is dead

if ((MyEnemy1HP > 0) && (MyEnemy2HP > 0)) { DistanceEnemy1ToEnemy2 = Math.Abs(colNum - SecEnemyColNum) + Math.Abs(rowNum - SecEnemyRowNum); }

// MySoldiers Target.. who is closest?

if (DistanceToEnemy1 <= DistanceToEnemy2)

{

TargetColNum = EnemyColNum;

TargetRowNum = EnemyRowNum;

}

else

{

TargetColNum = SecEnemyColNum;

TargetRowNum = SecEnemyRowNum;

}

// Enemy1's target... soldier or enemy2

if (DistanceToEnemy1 <= DistanceEnemy1ToEnemy2)

{

Enemy1TargetColNum = colNum;

Enemy1TargetRowNum = rowNum;

}

else

{

Enemy1TargetColNum = SecEnemyColNum;

Enemy1TargetRowNum = SecEnemyRowNum;

}

// Enemy2's target... soldier or enemy1

if (DistanceToEnemy2 <= DistanceEnemy1ToEnemy2)

{

Enemy2TargetColNum = colNum;

Enemy2TargetRowNum = rowNum;

}

else

{

Enemy2TargetColNum = EnemyColNum;

Enemy2TargetRowNum = EnemyRowNum;

}

// Direction of ATTACK

//Decide which direction my soldier should go to, to attack his target

MySoldiersDirection = WhichDirectionToAttack(colNum, rowNum, TargetColNum, TargetRowNum);

//Decide which direction enemy1 should go to, to attack his target

MyEnemySoldiersDirection = WhichDirectionToAttack(EnemyColNum, EnemyRowNum, Enemy1TargetColNum, Enemy1TargetRowNum);

//Decide which direction enemy2 should go to, to attack his target

MySecEnemySoldiersDirection = WhichDirectionToAttack(SecEnemyColNum, SecEnemyRowNum, Enemy2TargetColNum, Enemy2TargetRowNum);

//Direction of RETREAT... if alive but under 5 healthpoints

if (MySoldiersHP > 0 && MySoldiersHP < 5) { MySoldiersDirection = WhichDirectionToRetreat(colNum, rowNum, TargetColNum, TargetRowNum); }

if (MyEnemy1HP > 0 && MyEnemy1HP < 5) { MyEnemySoldiersDirection = WhichDirectionToRetreat(Enemy1TargetColNum, Enemy1TargetRowNum, EnemyColNum, EnemyRowNum); }

if (MyEnemy2HP > 0 && MyEnemy2HP < 5) { MySecEnemySoldiersDirection = WhichDirectionToRetreat(Enemy2TargetColNum, Enemy2TargetRowNum, SecEnemyColNum, SecEnemyRowNum); }

/\*\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\*/

// Corner Escaping

// If Soldier is alive, check if he is in a corner, or is busy getting out of a corner... over-ride his direction here

if (MySoldiersHP > 0)

{

if ((colNum == 0 && rowNum == 0) || (colNum == 0 && rowNum == 19) || (colNum == 19 && rowNum == 0) || (colNum == 19 && rowNum == 19)

|| (SoldierCornerEscapeCounter > 0))

{

MySoldiersDirection = CornerEscapeDirection(TargetColNum, TargetRowNum, colNum, rowNum,

ref SoldierCornerLabel, ref SoldierCornerEscape1stDirection, ref SoldierCornerEscapeCounter);

}

}

// If Enemy1 is alive, check if Enemy1 is in a corner, or is busy getting out of a corner... over-ride his direction here

if (MyEnemy1HP > 0)

{

if ((EnemyColNum == 0 && EnemyRowNum == 0) || (EnemyColNum == 0 && EnemyRowNum == 19) || (EnemyColNum == 19 && EnemyRowNum == 0) || (EnemyColNum == 19 && EnemyRowNum == 19)

|| (Enemy1CornerEscapeCounter > 0))

{

MyEnemySoldiersDirection = CornerEscapeDirection(Enemy1TargetColNum, Enemy1TargetRowNum, EnemyColNum, EnemyRowNum,

ref Enemy1CornerLabel, ref Enemy1CornerEscape1stDirection, ref Enemy1CornerEscapeCounter);

}

}

// If Enemy2 is alive, Check if Enemy2 is in a corner, or is busy getting out of a corner... over-ride his direction here

if (MyEnemy2HP > 0)

{

if ((SecEnemyColNum == 0 && SecEnemyRowNum == 0) || (SecEnemyColNum == 0 && SecEnemyRowNum == 19) || (SecEnemyColNum == 19 && SecEnemyRowNum == 0) || (SecEnemyColNum == 19 && SecEnemyRowNum == 19)

|| (Enemy2CornerEscapeCounter > 0))

{

MySecEnemySoldiersDirection = CornerEscapeDirection(Enemy2TargetColNum, Enemy2TargetRowNum, SecEnemyColNum, SecEnemyRowNum,

ref Enemy2CornerLabel, ref Enemy2CornerEscape1stDirection, ref Enemy2CornerEscapeCounter);

}

}

/\*\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\*/

// end of Corner Escaping

//MY SOLDIERS NEW CELL

if (MySoldiersHP > 0)

{

//move Right:

if (MySoldiersDirection == moveRight && colNum < 19) { colNum++; }

//move Left

if (MySoldiersDirection == moveLeft && colNum > 0) { colNum--; }

//move Up

if (MySoldiersDirection == moveUp && rowNum > 0) { rowNum--; }

//move Down

if (MySoldiersDirection == moveDown && rowNum < 19) { rowNum++; }

}

//MY ENEMYs new cell

if (MyEnemy1HP > 0)

{

//move Right:

if (MyEnemySoldiersDirection == moveRight && EnemyColNum < 19)

{

EnemyColNum++;

}

//move Left

else if (MyEnemySoldiersDirection == moveLeft && EnemyColNum > 0)

{

EnemyColNum--;

}

//move Up

else if (MyEnemySoldiersDirection == moveUp && EnemyRowNum > 0)

{

EnemyRowNum--;

}

//move Down

else if (MyEnemySoldiersDirection == moveDown && EnemyRowNum < 19)

{

EnemyRowNum++;

}

}

//MY SECOND ENEMY'S new CELL

if (MyEnemy2HP > 0)

{

//move Right:

if (MySecEnemySoldiersDirection == moveRight && SecEnemyColNum < 19)

{

SecEnemyColNum++;

}

//move Left

else if (MySecEnemySoldiersDirection == moveLeft && SecEnemyColNum > 0)

{

SecEnemyColNum--;

}

//move Up

else if (MySecEnemySoldiersDirection == moveUp && SecEnemyRowNum > 0)

{

SecEnemyRowNum--;

}

//move Down

else if (MySecEnemySoldiersDirection == moveDown && SecEnemyRowNum < 19)

{

SecEnemyRowNum++;

}

}

// display the images in their cells

MyGrid.Rows[rowNum].Cells[colNum].Value = MySoldierImg;

MyGrid.Rows[EnemyRowNum].Cells[EnemyColNum].Value = EnemySoldierImg;

MyGrid.Rows[SecEnemyRowNum].Cells[SecEnemyColNum].Value = SecEnemySoldierImg;

//ATTACK IF WITHIN RANGE!!!!!!!!!!!!!!!!!!!!!!!!!!!!

//(1) Check if my soldier is within attack range of enemy1... if both is alive

ATTACK(ref MySoldiersHP, colNum, rowNum, ref MyEnemy1HP, EnemyColNum, EnemyRowNum, ref EnemySoldierImg);

//(2)Check if my soldier is within attack range of enemy2... if both is alive

ATTACK(ref MySoldiersHP, colNum, rowNum, ref MyEnemy2HP, SecEnemyColNum, SecEnemyRowNum, ref SecEnemySoldierImg);

//(3)Check if enemy1 is within attack range of soldier ... if both is alive

ATTACK(ref MyEnemy1HP, EnemyColNum, EnemyRowNum, ref MySoldiersHP, colNum, rowNum, ref MySoldierImg);

//(4)Check if enemy1 is within attack range of enemy2 ... if both is alive

ATTACK(ref MyEnemy1HP, EnemyColNum, EnemyRowNum, ref MyEnemy2HP, SecEnemyColNum, SecEnemyRowNum, ref SecEnemySoldierImg);

//(5)Check if enemy2 is within attack range of enemy1 ... if both is alive

ATTACK(ref MyEnemy2HP, SecEnemyColNum, SecEnemyRowNum, ref MyEnemy1HP, EnemyColNum, EnemyRowNum, ref EnemySoldierImg);

//(6)Check if enemy2 is within attack range of soldier ... if both is alive

ATTACK(ref MyEnemy2HP, SecEnemyColNum, SecEnemyRowNum, ref MySoldiersHP, colNum, rowNum, ref MySoldierImg);

}

// end of GameTimerTick

// this function decides which direction I must advance toward an enemy, given my FROM position and the TO position of the enemy

public int WhichDirectionToAdvance(int FromPosX, int FromPosY, int ToPosX, int ToPosY, int MyAttackRange)

{ // returns 1,2,3 or 4.... or 0 (stop) if the distance is within my attackrange

// Note: FROM is me, the advancing unit …. and TO is the enemy I must go towards

//Deciding to move horizontally or vertically?... choose the greatest distance

int HorizontalDif, VerticalDif, Direction;

bool MoveHorizontally;

HorizontalDif = Math.Abs(FromPosX - ToPosX);

VerticalDif = Math.Abs(FromPosY - ToPosY);

if ((HorizontalDif > VerticalDif) && ((FromPosX != 0) | (FromPosX != 19)))

{

MoveHorizontally = true; //true means horizontal

}

else if ((HorizontalDif < VerticalDif) && ((FromPosY != 0) | (FromPosY != 19)))

{

MoveHorizontally = false; //false means vertical

}

else { MoveHorizontally = true; }

if (MoveHorizontally == true) //Horizontal movement

{

if (FromPosX < ToPosX)

{

Direction = 1; //Right

}

else

{

Direction = 2; //left

}

}

else //Vertical Movement

{

if (FromPosY > ToPosY)

{

Direction = 3; //Up

}

else

{

Direction = 4; //Down

}

}

if (HorizontalDif <= AttackRange && VerticalDif <= AttackRange) // within attack range ..therefore stop and KILL!

{

Direction = 0;

}

return Direction;

} // end of WHICHDIRECTIONTOADVANCE

public int WhichDirectionToRetreat(int FromPosX, int FromPosY, int ToPosX, int ToPosY)

{ // returns 1,2,3 or 4.... TO is ‘me’ running away from the 'FROM' attacker

//Deciding to move horizontally or vertically away?... choose the smaller distance

int HorizontalDif, VerticalDif, Direction;

bool MoveHorizontally;

HorizontalDif = Math.Abs(FromPosX - ToPosX);

VerticalDif = Math.Abs(FromPosY - ToPosY);

// Plan A is to retreat along the shortest direction

if (HorizontalDif >= VerticalDif)

//if ((HorizontalDif >= VerticalDif) && ((ToPosY != 0) && (ToPosY != 19)))

{

MoveHorizontally = false; //false means vertical

}

else //if ((HorizontalDif < VerticalDif) && ((ToPosX != 0) | (ToPosX != 19)))

{

MoveHorizontally = true; //true means horizontal

}

//else { MoveHorizontally = false; } //this is not actually needed

if (MoveHorizontally == true) //Horizontal movement

{ // right or left

if (FromPosX <= ToPosX)

{

Direction = 1; //Right

}

else

{

Direction = 2; //left

}

}

else //vertical movement .... up or down?

{

if (FromPosY >= ToPosY)

{

Direction = 3; //Up

}

else

{

Direction = 4; //Down

}

}

//Plan B ... THIS SECTION HANDLES what to do if you have hit the border

//THIS IS MY VERTICAL MOVEMENT FOR me towards THE BORDER but then I change direction to Horizomntal along the border

if (MoveHorizontally == false)

{

if ((Direction == 3) && (ToPosY == 0)) // up but cannot go up!

{

if (FromPosX >= ToPosX)

{

Direction = 2; // rather go left along the border

}

else

{

Direction = 1; //rather go right along the border

}

}

if ((Direction == 4) && (ToPosY == 19)) // DOWN, but cannot go down

{

if (FromPosX >= ToPosX)

{

Direction = 2; // rather go left along the border

}

else

{

Direction = 1; //rather go right along the border

}

}

}

//THIS IS MY HORIZONTAL MOVEMENT FOR me towards THE BORDER but then I change direction to Vertical along the border

if (MoveHorizontally == true)

{

if ((Direction == 1) && (ToPosX == 19)) // right but cannot go right !

{

if (FromPosY >= ToPosY)

{

Direction = 3; // rather go UP along the border

}

else

{

Direction = 4; //rather go DOWN along the border

}

}

if ((Direction == 2) && (ToPosX == 0)) // left, but cannot go left

{

if (FromPosY >= ToPosY)

{

Direction = 3; // rather go UP along the border

}

else

{

Direction = 4; //rather go DOWN along the border

}

}

}

// finally return the direction

return Direction;

} // END OF FUNCTION - WHICH DIRECTION TO RETREAT

//CORNER ESCAPE:

public int CornerEscapeDirection(int FromPosX, int FromPosY, int ToPosX, int ToPosY,

ref string CornerLabel, ref int CornerEscape1stDirection, ref int CornerCounter)

{

// find the horizontal and vertical difference between me ('To') and my attacker ('From')

int HorizontalDif, VerticalDif;

HorizontalDif = Math.Abs(FromPosX - ToPosX);

VerticalDif = Math.Abs(FromPosY - ToPosY);

//Step 1: Increment Corner Counter (passed in as a parameter) and set direction to 0..ie dont move !

CornerCounter++;

int Direction = 0;

//Step 2: Set Corner Label on the first corner escape iteration.

//This is a 'by ref' paremeter i.e: it is 'global'

if (CornerCounter == 1)

{

if (ToPosX == 0 && ToPosY == 0) { CornerLabel = "TopLeft"; }

if (ToPosX == 19 && ToPosY == 0) { CornerLabel = "TopRight"; }

if (ToPosX == 0 && ToPosY == 19) { CornerLabel = "BottomLeft"; }

if (ToPosX == 19 && ToPosY == 19) { CornerLabel = "BottomRight"; }

Direction = 0; //Do not move

return Direction; //exit!

}

//Step 3: If corner counter is 2 or 3 simply exit... dont move

if (CornerCounter == 2 || CornerCounter == 3)

{

Direction = 0; //Do not move

return Direction; //exit!

}

//Step 4: If corner counter is 4, then start ESCAPING!!! ...horizontally or vertically away from the corner... choose the smaller distance

if (CornerCounter == 4)

{

//Code for Top Left Corner

if (CornerLabel.Equals("TopLeft"))

{

CornerEscape1stDirection = 1; //right

if (HorizontalDif > VerticalDif) { CornerEscape1stDirection = 4; } //Down

}

//Code for Top Right Corner

if (CornerLabel.Equals("TopRight"))

{

CornerEscape1stDirection = 2; //left

if (HorizontalDif > VerticalDif) { CornerEscape1stDirection = 4; } //Down

}

//Code for Bottom Left Corner

if (CornerLabel.Equals("BottomLeft"))

{

CornerEscape1stDirection = 1; //right

if (HorizontalDif > VerticalDif) { CornerEscape1stDirection = 3; } //up

}

//Code for Bottom Right Corner

if (CornerLabel.Equals("BottomRight"))

{

CornerEscape1stDirection = 2; //left

if (HorizontalDif > VerticalDif) { CornerEscape1stDirection = 3; } //up

}

Direction = CornerEscape1stDirection;

return Direction; //exit!

} // end of Step 4

//Step 5: Keep Going In The 1st Direction chosen....between moves 5 and 8

if (CornerCounter >= 5 && CornerCounter <= 8 )

{

Direction = CornerEscape1stDirection;

return Direction; //exit!

}

//Step 6: If corner counter > 8 then move in a perpendicular direction for 4 iterations

if (CornerCounter >= 9 && CornerCounter <= 12)

{

//Code for Top Left Corner

if (CornerLabel.Equals("TopLeft") && CornerEscape1stDirection == 1) { Direction = 4; } //down

if (CornerLabel.Equals("TopLeft") && CornerEscape1stDirection == 4) { Direction = 1; } //right

//Code for Top Right Corner

if (CornerLabel.Equals("TopRight") && CornerEscape1stDirection == 2) { Direction = 4; } //down

if (CornerLabel.Equals("TopRight") && CornerEscape1stDirection == 4) { Direction = 2; } //left

//Code for Bottom Right Corner

if (CornerLabel.Equals("BottomRight") && CornerEscape1stDirection == 2) { Direction = 3; } //up

if (CornerLabel.Equals("BottomRigh") && CornerEscape1stDirection == 3) { Direction = 2; } //left

//Code for Bottom Left Corner

if (CornerLabel.Equals("BottomLeft") && CornerEscape1stDirection == 1) { Direction = 3; } //up

if (CornerLabel.Equals("BottomLeft") && CornerEscape1stDirection == 3) { Direction = 1; } //right

return Direction; //exit!

} // end of Step 6

// after the 12th move set the corner counter of this enemy back to zero

if (CornerCounter > 12) { CornerCounter = 0; }

//default exit

return Direction;

} // end of function CornerEscapDirection

public void ATTACK(ref int Ahp, int Acol, int Arow, ref int Bhp, int Bcol, int Brow, ref Image Bimage)

{ // A attacks B, reducing his HP... and blanking his image if he (B) is dead

//Check if A is within attack range of B... and both is alive

if (Ahp > 0 && Bhp > 0)

{

if ((Math.Abs(Acol - Bcol) <= AttackRange) && (Math.Abs(Arow - Brow) <= AttackRange))

{

// attack!!...reduce B's health points

Bhp--;

Console.Beep(2000, 100); //quick sound

//check if B has died... blank out image...long gunshot!

if (Bhp == 0)

{

Bimage = null;

Console.Beep(800, 500); //long sound

}

}

}

}//end of ATTACK code

}//end of Form1

}