using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

using System.Media; // to play sounf efx!

using System.IO;

namespace WindowsFormsApplication1

{

public partial class MainForm : Form

{

public MainForm()

{

InitializeComponent();

}

//Global variables

public static SoundPlayer MySoundPlayer = new SoundPlayer();

public static int RoundNumber;

public static int moveRight = 1;

public static int moveLeft = 2;

public static int moveUp = 3;

public static int moveDown = 4;

public static MeleeUnit[] MyArrayOfMeleeUnits;

public static RangedUnit[] MyArrayOfRangedUnits;

public static int NumberOfMeleeUnits;

public static int NumberOfRangedUnits;

public static bool SoundHasAlreadyPlayed = false; // every round, only one unit can get a sound effect when attacking

// global variable of folder to use to create csv files… NOTE!!! Please ensure the pc has this folder !!!!

public static string FileFolder = "C:\\Temp";

//Random Number Generator:

public static Random randomNumberGenerator = new Random();

// variables for BUILDINGS

public static ResourceBuilding [] MyArrayOfResourceBuildings;

public static FactoryBuilding[] MyArrayOfFactoryBuildings;

public static int NumberOfResourceBuildings;

public static int NumberOfFactoryBuildings;

//-----Question 1.7a----- Create a map...using a datagridview...

//Create a customized DataGridView which is transparent (so as to display its 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 = RTS\_Game\_Project\_18000551\_Jesse\_Hiebner.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...to make it transparent... 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.DrawImage(this.MyBackgroundImage, gridBounds); //...and then draw MyBackgroundImage (property created above) over 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 transparent grid with picture... created in memory

// now declare MyGrid as an instance of the transparent/picture grid

public static MyTransparentDataGrid MyGrid = new MyTransparentDataGrid();

// ...and add columns/rows to MyGrid in this method SettingPropertiesForMyGrid

public void SettingPropertiesForMyGrid()

{// ...add columns/rows to 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(90, 90); // top left corner position

MyGrid.Size = new System.Drawing.Size(510, 510); // width, height

//Making the grid lines transparent.

MyGrid.CellBorderStyle = DataGridViewCellBorderStyle.None;

//Clear the 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;

}// end of setting properties for my grid

// load the MainForm and do initial settings

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

{

//Form size

Width = 1600;

Height = 800;

//startup sounds !

MySoundPlayer.Stream = RTS\_Game\_Project\_18000551\_Jesse\_Hiebner.Properties.Resources.Start\_of\_Play;

MySoundPlayer.Play();

// add rows/columns to MyGrid:

SettingPropertiesForMyGrid();

// ...and now add the map (MyGrid) to the form

Controls.Add(MyGrid);

//starting comment in Round Number label

lblRoundNumber.Text = "...game has not yet started";

} // end Main Form load

// buttons --------------------------------------------------------------------------------------------

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

{

Close();

}

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

{

//sound efx background music

MySoundPlayer.Stream = RTS\_Game\_Project\_18000551\_Jesse\_Hiebner.Properties.Resources.GunShot;

MySoundPlayer.Play();

// start the timer

GameTimer.Start();

}

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

{

GameTimer.Stop();

}

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

{ // a button to create and display units on the map before the game begins

// also create buildings and display them on the map

//sound efx

MySoundPlayer.Stream = RTS\_Game\_Project\_18000551\_Jesse\_Hiebner.Properties.Resources.GunShot;

MySoundPlayer.Play();

//clear the winner label (from the previous game session)

lblWinner.Text = "";

// creating a new 'map' .... instantiates a new Map... decide how many melee vs ranged (eg 2 & 1) ... and how many buildings

int n = randomNumberGenerator.Next(5, 11); //generate a random amount of units between 5 and 10

int n2 = randomNumberGenerator.Next(3, 7); //generate a random total amount of buildings to create …. between 3 and 6

Map MyMap = new Map(n,n2); // instantiate a new MAP with n amount of units, and n2 amount of buildings. (note: NumberOfMeleeUnits & NumberOfRangedUnits will be set here)

//instantiate the global arrays ...for easy referencing

MeleeUnit[] localMyArrayOfMeleeUnits = new MeleeUnit[NumberOfMeleeUnits];

RangedUnit[] localMyArrayOfRangedUnits = new RangedUnit[NumberOfRangedUnits];

ResourceBuilding[] localMyArrayOfResourceBuildings = new ResourceBuilding[NumberOfResourceBuildings];

FactoryBuilding[] localMyArrayOfFactoryBuildings = new FactoryBuilding[NumberOfFactoryBuildings];

// generate the units randomly... and store them in the 2 global arrays ...do the same for buildings

Map.GenerateUnits(ref localMyArrayOfMeleeUnits, ref localMyArrayOfRangedUnits);

Map.GenerateBuildings(ref localMyArrayOfResourceBuildings, ref localMyArrayOfFactoryBuildings);

//Display units on the map, as well as populating the textbox report on the richtextbox 'RTB'….do the same for BUILDINGS!

MainForm.Map.DisplayAllUnits(localMyArrayOfMeleeUnits, localMyArrayOfRangedUnits, RTB);

MainForm.Map.DisplayAllBuildings(localMyArrayOfResourceBuildings, localMyArrayOfFactoryBuildings, RTB);

MyArrayOfMeleeUnits = localMyArrayOfMeleeUnits;

MyArrayOfRangedUnits = localMyArrayOfRangedUnits;

MyArrayOfResourceBuildings = localMyArrayOfResourceBuildings;

MyArrayOfFactoryBuildings = localMyArrayOfFactoryBuildings;

//play and pause button are only enabled after units have been created on screen:

btnPlay.Enabled = true;

btnPause.Enabled = true;

}// end of CreateClick

// Question 2 - SAVE button

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

{

} // dummy unused SAVE ....problems when removing it!

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

{

// write to a csv file…all units & building info

var CompleteString = new StringBuilder();

string newline;

string FileName;

//clear string from previous session:

// Step 1 - write Melee units

CompleteString.Clear();

FileName = FileFolder + "\\GameSettingsMeleeUnits.CSV";

File.WriteAllText(FileName, String.Empty);

File.Create(FileName).Close(); //works

for (int i = 0; i < MyArrayOfMeleeUnits.Length; i++)

{

newline = MyArrayOfMeleeUnits[i].SaveSettingsAsString(); //calls the Save string method

CompleteString.AppendLine(newline); // append the line of data to the StringBuilder

}

// put the entire string into the file

File.AppendAllText(FileName, CompleteString.ToString());

FileName = "";

// Step 2 - write Ranged units to csv file

CompleteString.Clear();

FileName = FileFolder + "\\GameSettingsRangedUnits.CSV";

File.WriteAllText(FileName, String.Empty);

File.Create(FileName).Close(); //create/over-write a file

for (int i = 0; i < MyArrayOfRangedUnits.Length; i++)

{

newline = MyArrayOfRangedUnits[i].SaveSettingsAsString(); //calls the Save string method

CompleteString.AppendLine(newline); // append the line of data to the StringBuilder

}

// put the entire string into the file

File.AppendAllText(FileName, CompleteString.ToString());

// Step 3 - write Resource Buidings units to csv file

CompleteString.Clear();

FileName = FileFolder + "\\GameSettingsResourceBuildings.CSV";

File.WriteAllText(FileName, String.Empty);

File.Create(FileName).Close(); ; //create/over-write a file

for (int i = 0; i < MyArrayOfResourceBuildings.Length; i++)

{

newline = MyArrayOfResourceBuildings[i].SaveSettingsAsString(); //calls the Save string method

CompleteString.AppendLine(newline); // append the line of data to the StringBuilder

}

// put the entire string into the file

File.AppendAllText(FileName, CompleteString.ToString());

// Step 4 - write Factory Buidings to csv file

CompleteString.Clear();

FileName = FileFolder + "\\GameSettingsFactoryBuildings.CSV";

File.WriteAllText(FileName, String.Empty);

File.Create(FileName).Close(); //create/over-write a file

for (int i = 0; i < MyArrayOfFactoryBuildings.Length; i++)

{

newline = MyArrayOfFactoryBuildings[i].SaveSettingsAsString(); //calls the Save string method

CompleteString.AppendLine(newline); // append the line of data to the StringBuilder

}

// put the entire string into the file

File.AppendAllText(FileName, CompleteString.ToString());

} // end of SAVE btn

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

{

//reads all csv filed…all units & building info ….and store it in the arrays… and finally displays everything (in the arrays) on the grid

Map.ReadFilesAndDisplay(RTB);

}

public static string ConvertImageToString(Image MyImage)

{ // this method converts an image to a string of bytes…to save it to a file…. help obtained from: http://a-sidahmed.blogspot.com/2013/05/c-convertir-image-en-string-et-string.html

// …uses MemoryStream

if (MyImage == null) { return String.Empty; }

var MyMemoryStream = new MemoryStream();

MyImage.Save(MyMemoryStream, MyImage.RawFormat);

var MyMemoryStreamArray = MyMemoryStream.ToArray();

return Convert.ToBase64String(MyMemoryStreamArray);

} // end of Convert Image to String

public static Image ConvertStringToImage(string MyImageString)

{

if (String.IsNullOrWhiteSpace(MyImageString)) { return null; }

var MyImageBytes = Convert.FromBase64String(MyImageString);

var MyImageStream = new MemoryStream(MyImageBytes); //uses memorysteam

return Image.FromStream(MyImageStream);

}

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

{

} // end of READ button

// GAME TIMER ----------------------------------------------------------------------------------

// Game Time Tick event ...every second

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

{

// This method works as follows:

// Step 1 - display the round counter

// Step 2 - Check if there is a winning team. If so then display a winning message and pause the game (a team wins if all the opponents are dead)

// Step 3 - Check if Melees can move this round - based on their speed/slowness factor - use the MODULUS function

// Step 4 - If Melees can move, then loop thru all Melee units, and for each unit, that is still alive, do as follows...

// a. find my nearest enemy unit

// b. decide if I must advance, retreat, or attack

// c. if I must advance then in what direction? or if I must retreat then in what direction?

// d. if i must attack then attack

// Step 5 - Check if Ranged units can move this round - based on their speed/slowness factor - use the MODULUS function

// Step 6 - Do the above Step 4 for Ranged units

// Step 7 - Finally, display all units on the map.... call MAP's DisplayAllUnits method...

// Step 8 – Loop thru all ResourceBuildings…generate resources (if there are left in its pool)

// Step 9 – Loop thru all FactoryBuildings…check if it can spawn a new unit in this round? (use Modulus operator)…. If so then spawn a new unit….

// Step 10 – Finally, display all buildings…....( call MAP's DisplayAllBuildings) .. as well as their info in the richtextbox

string MyTeam;

int EnemyX, EnemyY, EnemyArrayIndex, Direction;

string EnemyUnitType;

// Step 1 - display the round counter

RoundNumber = RoundNumber + 1;

lblRoundNumber.Text = RoundNumber.ToString();

SoundHasAlreadyPlayed = false; // set the boolean to false

// Step 2 - Check if there is a winning team. If so then display a winning message and pause the game (a team wins if all the opponents are dead)

if (GameEngine.IsAllBravoDead())

{

//sound efx - victory celebration

MySoundPlayer.Stream = RTS\_Game\_Project\_18000551\_Jesse\_Hiebner.Properties.Resources.CanonShot;

MySoundPlayer.Play();

lblWinner.Text = "ALPHA WINS!";

GameTimer.Stop();

return; //exit

}

if (GameEngine.IsAllAlphaDead())

{

//sound efx - victory celebration

MySoundPlayer.Stream = RTS\_Game\_Project\_18000551\_Jesse\_Hiebner.Properties.Resources.CanonShot;

MySoundPlayer.Play();

lblWinner.Text = "BRAVO WINS!";

GameTimer.Stop();

return; //exit

}

// Step 3 - Check if Melees can move this round - based on their speed/slowness factor - use the MODULUS function

if ((RoundNumber % MyArrayOfMeleeUnits[0].Speed) == 0) // use the MODULUS operator

{

// Step 4 - Loop thru all Melee units, and for each unit that is still alive (health points is positive) do as follows...

// a. find my nearest enemy unit

// b. decide if I must advance, retreat, or attack...and act accordingly

// c. if I must advance then in what direction? or if I must retreat then in what direction?

//Loop thru all Melee units, and for each living unit, do as follows...

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

{

//reset the IsAttacking field back to false

MyArrayOfMeleeUnits[i].IsAttacking = false;

//am I still alive?

if (MyArrayOfMeleeUnits[i].Health > 0)

{

MyTeam = MyArrayOfMeleeUnits[i].Faction;

//a. find my nearest living enemy unit or building ... the method will populate the following 4 variables

EnemyX = 0;

EnemyY = 0;

EnemyArrayIndex = 0;

EnemyUnitType = "";

MyArrayOfMeleeUnits[i].PositionOfNearestEnemyUnitOrBuilding(ref EnemyX, ref EnemyY, ref EnemyUnitType, ref EnemyArrayIndex); // calls method on the Melee unit

// b. decide if I must retreat, advance, or attack ?...and acts accordingly

//If I'm injured, retreat, but only if the enemy type is melee or ranged! (i.e. attack buildings even if Im unhealthy, because buildings dont attack back)

if ((MyArrayOfMeleeUnits[i].Health < (0.25 \* MyArrayOfMeleeUnits[i].MaxHealth)) && ( EnemyUnitType.Equals("Melee")||EnemyUnitType.Equals("Ranged") ) )

{

// retreat

Direction = GameEngine.WhichDirectionToRetreat(EnemyX, EnemyY, MyArrayOfMeleeUnits[i].X, MyArrayOfMeleeUnits[i].Y);

MyArrayOfMeleeUnits[i].MoveToNewPosition(Direction); // calls method on the Melee unit

}

else // I'm healthy!...so I can advance or attack

{

//Check if I am in within attack range so I can attack

if ((Math.Abs(MyArrayOfMeleeUnits[i].X - EnemyX) <= MyArrayOfMeleeUnits[i].AttackRange) && (Math.Abs(MyArrayOfMeleeUnits[i].Y - EnemyY) <= MyArrayOfMeleeUnits[i].AttackRange))

{

// "Attack";

MyArrayOfMeleeUnits[i].HandleCombatWithEnemy(EnemyUnitType, EnemyArrayIndex); // calls method on the Melee unit

Direction = 0;

if (SoundHasAlreadyPlayed == false)

{

//sound efx

MySoundPlayer.Stream = RTS\_Game\_Project\_18000551\_Jesse\_Hiebner.Properties.Resources.Punch; // Melee's punch sounds like a glass smash!

MySoundPlayer.Play();

SoundHasAlreadyPlayed = true;

}

}

else // i will advance

{

// Advance";

Direction = GameEngine.WhichDirectionToAdvance(MyArrayOfMeleeUnits[i].X, MyArrayOfMeleeUnits[i].Y, EnemyX, EnemyY, MyArrayOfMeleeUnits[i].AttackRange);

MyArrayOfMeleeUnits[i].MoveToNewPosition(Direction); // calls method on the Melee unit

}

} // end of action to take

}// I'm alive

// I am dead!.... (my Health is <= 0)

else

{

// call the HandleMyDeath method... which replaces the Symbol with a 'dead' symbol

MyArrayOfMeleeUnits[i].MyDeath(i);

}

}// end of Melee loop

}// Melees can move in this round

// now do the same for the Ranged Units -----------------------------------------------------

// Step 5 - Check if Ranged can move this round - based on their speed/slowness factor - use the MODULUS function

if ((RoundNumber % MyArrayOfRangedUnits[0].Speed) == 0) // use the MODULUS operator

{

//Step 6 - Loop thru all Ranged units, and for each living unit, do as follows...

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

{

//reset the IsAttacking field back to false

MyArrayOfRangedUnits[i].IsAttacking = false;

//am I still alive?

if (MyArrayOfRangedUnits[i].Health > 0)

{

MyTeam = MyArrayOfRangedUnits[i].Faction;

//a. find my nearest living enemy unit or building... the method will populate the following 4 variables

EnemyX = 0;

EnemyY = 0;

EnemyArrayIndex = 0;

EnemyUnitType = "";

MyArrayOfRangedUnits[i].PositionOfNearestEnemyUnitOrBuilding(ref EnemyX, ref EnemyY, ref EnemyUnitType, ref EnemyArrayIndex); // calls method on the Ranged unit

// b. decide if I must retreat, advance, or attack ?...and acts accordingly

///If I'm injured, retreat,..... but only if the enemy type is melee or ranged! (i.e. attack buildings even if Im unhealthy, because buildings dont attack back)

if ((MyArrayOfRangedUnits[i].Health < (0.25 \* MyArrayOfRangedUnits[i].MaxHealth)) && (EnemyUnitType.Equals("Melee") || EnemyUnitType.Equals("Ranged")))

{

// "Retreat";

Direction = GameEngine.WhichDirectionToRetreat(EnemyX, EnemyY, MyArrayOfRangedUnits[i].X, MyArrayOfRangedUnits[i].Y);

MyArrayOfRangedUnits[i].MoveToNewPosition(Direction); // calls method on the Ranged unit

}

else // attack or advance

{

if ((Math.Abs(MyArrayOfRangedUnits[i].X - EnemyX) <= MyArrayOfRangedUnits[i].AttackRange) && (Math.Abs(MyArrayOfRangedUnits[i].Y - EnemyY) <= MyArrayOfRangedUnits[i].AttackRange))

{

// "Attack";

MyArrayOfRangedUnits[i].HandleCombatWithEnemy(EnemyUnitType, EnemyArrayIndex); // calls method on the Ranged unit

Direction = 0;

if(SoundHasAlreadyPlayed == false)

{

//sound efx

MySoundPlayer.Stream = RTS\_Game\_Project\_18000551\_Jesse\_Hiebner.Properties.Resources.LaserShot; // Ranged Unit has a laser gun !

MySoundPlayer.Play();

SoundHasAlreadyPlayed = true;

}

}

else

{

// "Advance";

Direction = GameEngine.WhichDirectionToAdvance(MyArrayOfRangedUnits[i].X, MyArrayOfRangedUnits[i].Y, EnemyX, EnemyY, MyArrayOfRangedUnits[i].AttackRange);

MyArrayOfRangedUnits[i].MoveToNewPosition(Direction); // calls method on the Melee unit

}

} // end of action to take

}// I'm alive

else //I am dead !!

{

// call the MyDeath method... which replaces the Symbol with a 'dead' symbol

MyArrayOfRangedUnits[i].MyDeath(i);

}

} // end of Ranged loop

}// Ranged can move in this round

//Step 7 ... Display all units on the map.... call MAP's DisplayAllUnits method...

// ...as well as populating the richtextbox report 'RTB'

Map.DisplayAllUnits(MyArrayOfMeleeUnits, MyArrayOfRangedUnits, RTB);

// Step 8 – Loop thru all ResourceBuildings…generate resources (if there are left in its pool )…

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

{ MyArrayOfResourceBuildings[i].GenerateResourcesFromThePool(); } // end of loop – resource buildings

// Step 9 – Loop thru all FactoryBuildings…check if they can spawn new units in this round? (use Modulus operator)…. If so then spawn a unit

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

{

if ((RoundNumber % MyArrayOfFactoryBuildings[i].ProductionSpeed) == 0) // use the MODULUS operator between RoundNumber and ProductionSpeed

{ MyArrayOfFactoryBuildings[i].SpawnAnotherUnit(); }

} // end of loop –factory buildings

// Step 10 – Finally, display all buildings…....( call MAP's DisplayAllBuildings) .. as well as their info in the richtextbox

Map.DisplayAllBuildings(MyArrayOfResourceBuildings, MyArrayOfFactoryBuildings, RTB);

} // end of game timer tick event -------------------------------------------------

////------------- PART 2 \_\_ Class Hierarchy------------------------------------------------------

//This is the parent base class UNIT

public abstract class Unit

{

protected int X;

protected int Y;

protected int Health;

protected int MaxHealth;

protected int Speed;

protected int Attack;

protected int AttackRange;

protected string Faction;

protected Image Symbol;

protected bool IsAttacking;

// Question 2 - add Name property

protected string Name;

//Question 1.4 ----------Constructor for parent base UNIT ----------

public Unit(int paramX, int paramY, int paramHealth, int paramSpeed, int paramAttack,

int paramAttackRange, string paramFaction, Image paramSymbol, bool paramIsAttacking)

{

X = paramX;

Y = paramY;

Health = paramHealth;

Speed = paramSpeed;

Attack = paramAttack;

AttackRange = paramAttackRange;

Faction = paramFaction;

Symbol = paramSymbol;

IsAttacking = paramIsAttacking;

}

//Question 1.4 ---- abstract methods which will be overridden:

public abstract void MoveToNewPosition(int MoveDirection);

public abstract void HandleCombatWithEnemy(string EnemyUnitType, int EnemyArrayIndex);

public abstract bool WithinRangeOfEnemy(string EnemyType, MeleeUnit MeleeEnemy, RangedUnit RangedEnemy);

public abstract void PositionOfNearestEnemyUnitOrBuilding(ref int PosX, ref int PoxY, ref string UnitType, ref int ArrayIndex);

public abstract void MyDeath(int ArrayIndex);

public abstract override string ToString();

//Question 2.11 ---- SAVE abstract method which will be overridden:

public abstract string SaveSettingsAsString();

} // end of UNIT class

//Question 1.5 ----- Creating child classes MeleeUnit and RangedUnit

//Beginning of Melee unit ------------------------------------------------------------------------------------

public class MeleeUnit : Unit

{

// Question 1.6 Accessors

private int accessor\_X;

public new int X { get { return accessor\_X; } set { accessor\_X = value; } }

private int accessor\_Y;

public new int Y { get { return accessor\_Y; } set { accessor\_Y = value; } }

private int accessor\_Health;

public new int Health { get { return accessor\_Health; } set { accessor\_Health = value; } }

// private int accessor\_MaxHealth;

public new int MaxHealth { get; } // accessor\_MaxHealth; } } //only needs a Get

private int accessor\_Speed;

public new int Speed { get { return accessor\_Speed; } set { accessor\_Speed = value; } }

private int accessor\_Attack;

public new int Attack { get { return accessor\_Attack; } set { accessor\_Attack = value; } }

private int accessor\_AttackRange;

public new int AttackRange { get { return accessor\_AttackRange; } set { accessor\_AttackRange = value; } }

private string accessor\_Faction;

public new string Faction {get { return accessor\_Faction; } set { accessor\_Faction = value; } }

private Image accessor\_Symbol;

public new Image Symbol { get { return accessor\_Symbol; } set { accessor\_Symbol = value; } }

private bool accessor\_IsAttacking;

public new bool IsAttacking { get { return accessor\_IsAttacking; } set { accessor\_IsAttacking = value; } }

//Question 2 - add Name property accessor

private string accessor\_Name;

public new string Name { get { return accessor\_Name; } set { accessor\_Name = value; } }

// constructor for Melee .... which calls the parent UNIT constructor

public MeleeUnit(int X, int Y, int Health, int Speed, int Attack, int AttackRange, string Faction, Image Symbol, bool IsAttacking)

: base(X, Y, Health, Speed, Attack, AttackRange, Faction, Symbol, IsAttacking)

{

// Question 1.5b ----- initialize MeleeUnit with relevant values: -----

this.X = randomNumberGenerator.Next(2, 19); // initial column postion between 2 and 19

this.Y = randomNumberGenerator.Next(2, 19); // initial row postion between 2 and 19

this.Health = 100; // initial Health set to 100

this.MaxHealth = this.Health; // Max health is set to initial health

this.Speed = 1; // this is actually the ?slowness? of the unit (move every nth round)

this.Attack = 5; // this is the attack-damage done to healthpoints of whoever this unit attacks

this.AttackRange = 1; // the attack range is always 1 cell fro Melees

int r = randomNumberGenerator.Next(1, 3); //placing this unit in team Alpha or Bravo

if (r == 1)

{

this.Faction = "Alpha";

this.Symbol = RTS\_Game\_Project\_18000551\_Jesse\_Hiebner.Properties.Resources.MeleeAlpha;

}

else

{

this.Faction = "Bravo";

this.Symbol = RTS\_Game\_Project\_18000551\_Jesse\_Hiebner.Properties.Resources.MeleeBravo; // Bravo pictures have a black border

}

this.IsAttacking = false; // intitalize "is attacking" to false

// Question 2 - add Name property

r = randomNumberGenerator.Next(1, 4); //giving Melee one of 3 random 'punch' names

if (r == 1) { this.Name = "KickBoxer"; }

if (r == 2) { this.Name = "NinjaFist"; }

if (r == 3) { this.Name = "BigPuncher"; }

}

//Overridden Methods: Question 1.4

public override void MoveToNewPosition(int Move)

{// this method sets the new X,Y position of a unit depending on what Move direction it goes to

// ie: Move can be .... 1=right, 2=left, 3=up, 4=down

if (Move == moveRight)

{

X = X + 1;

if (X > 19) { X = 19; }

}

if (Move == moveLeft)

{

X = X - 1;

if (X < 0) { X = 0; }

}

if (Move == moveUp)

{

Y = Y - 1;

if (Y < 0) { Y = 0; }

}

if (Move == moveDown)

{

Y = Y + 1;

if (Y > 19) { Y = 19; }

}

}// end to MoveToNewPosition

public override void HandleCombatWithEnemy(string EnemyUnitType, int EnemyArrayIndex)

{// method to attack an enemy... decrease his health with my attack-damge points

// set the IsAttacking field

IsAttacking = true;

if (EnemyUnitType == "Melee")

{

MyArrayOfMeleeUnits[EnemyArrayIndex].Health = MyArrayOfMeleeUnits[EnemyArrayIndex].Health - Attack;

//Negative damage exception

if(MyArrayOfMeleeUnits[EnemyArrayIndex].Health < 0) { MyArrayOfMeleeUnits[EnemyArrayIndex].Health = 0;}

}

if (EnemyUnitType == "Ranged")

{

MyArrayOfRangedUnits[EnemyArrayIndex].Health = MyArrayOfRangedUnits[EnemyArrayIndex].Health - Attack;

//Negative damage exception

if (MyArrayOfRangedUnits[EnemyArrayIndex].Health < 0) { MyArrayOfRangedUnits[EnemyArrayIndex].Health = 0;}

}

if (EnemyUnitType == "Resource")

{

MyArrayOfResourceBuildings[EnemyArrayIndex].Health = MyArrayOfResourceBuildings[EnemyArrayIndex].Health - Attack;

//Negative damage exception

if (MyArrayOfResourceBuildings[EnemyArrayIndex].Health < 0) { MyArrayOfResourceBuildings[EnemyArrayIndex].Health = 0; }

}

if (EnemyUnitType == "Factory")

{

MyArrayOfFactoryBuildings[EnemyArrayIndex].Health = MyArrayOfFactoryBuildings[EnemyArrayIndex].Health - Attack;

//Negative damage exception

if (MyArrayOfFactoryBuildings[EnemyArrayIndex].Health < 0) { MyArrayOfFactoryBuildings[EnemyArrayIndex].Health = 0; }

}

} // end of HandleCombat

public override bool WithinRangeOfEnemy(string EnemyType, MeleeUnit MeleeEnemy, RangedUnit RangedEnemy)

{ // method to determine whether another unit is within attack range? returns true or false

bool withinRange = false;

if (EnemyType == "Melee")

{

if ((Math.Abs(X - MeleeEnemy.X) <= AttackRange) && (Math.Abs(Y - MeleeEnemy.Y) <= AttackRange))

{

withinRange = true;

}

}

if (EnemyType == "Ranged")

{

if ((Math.Abs(X - RangedEnemy.X) <= AttackRange) && (Math.Abs(Y - RangedEnemy.Y) <= AttackRange))

{

withinRange = true;

}

}

return withinRange;

} // end of withinRangeOfEnemy

public override void PositionOfNearestEnemyUnitOrBuilding(ref int PosX, ref int PosY, ref string EnemyUnitType, ref int ArrayIndex)

{ // method to return position of the closest living enemy unit to me - via reference output parameters

// eg: output parameters will be: 12, 4, "Melee", 2 And this means .... the 2nd Melee unit in the array...whose X,y position is 12,4

int NearestDistance = 100;

int ThisDistance;

string Enemy;

if (Faction == "Alpha") { Enemy = "Bravo"; } else { Enemy = "Alpha"; }

//loop thru the Melee units looking for the nearest enemy

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

{

if ((MyArrayOfMeleeUnits[i].Faction == Enemy) && (MyArrayOfMeleeUnits[i].Health > 0))

{

ThisDistance = Math.Abs(X - MyArrayOfMeleeUnits[i].X) + Math.Abs(Y - MyArrayOfMeleeUnits[i].Y);

if (ThisDistance < NearestDistance)

{

NearestDistance = ThisDistance;

PosX = MyArrayOfMeleeUnits[i].X;

PosY = MyArrayOfMeleeUnits[i].Y;

EnemyUnitType = "Melee";

ArrayIndex = i;

}

}

}

//loop thru the Ranged units looking for an even nearer enemy

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

{

if ((MyArrayOfRangedUnits[i].Faction == Enemy) && (MyArrayOfRangedUnits[i].Health > 0))

{

ThisDistance = Math.Abs(X - MyArrayOfRangedUnits[i].X) + Math.Abs(Y - MyArrayOfRangedUnits[i].Y);

if (ThisDistance < NearestDistance)

{

NearestDistance = ThisDistance;

PosX = MyArrayOfRangedUnits[i].X;

PosY = MyArrayOfRangedUnits[i].Y;

EnemyUnitType = "Ranged";

ArrayIndex = i;

}

}

}

// Question 3: allow units to attack enemy resource buildings as well

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

{

if ((MyArrayOfResourceBuildings[i].Faction == Enemy) && (MyArrayOfResourceBuildings[i].Health > 0))

{

ThisDistance = Math.Abs(X - MyArrayOfResourceBuildings[i].X) + Math.Abs(Y - MyArrayOfResourceBuildings[i].Y);

if (ThisDistance < NearestDistance)

{

NearestDistance = ThisDistance;

PosX = MyArrayOfResourceBuildings[i].X;

PosY = MyArrayOfResourceBuildings[i].Y;

EnemyUnitType = "Resource";

ArrayIndex = i;

}

}

}

// Question 3: allow units to attack enemy Factory buildings as well

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

{

if ((MyArrayOfFactoryBuildings[i].Faction == Enemy) && (MyArrayOfFactoryBuildings[i].Health > 0))

{

ThisDistance = Math.Abs(X - MyArrayOfFactoryBuildings[i].X) + Math.Abs(Y - MyArrayOfFactoryBuildings[i].Y);

if (ThisDistance < NearestDistance)

{

NearestDistance = ThisDistance;

PosX = MyArrayOfFactoryBuildings[i].X;

PosY = MyArrayOfFactoryBuildings[i].Y;

EnemyUnitType = "Factory";

ArrayIndex = i;

}

}

}

return;

} // end of Position of Nearest Enemy Unit

public override void MyDeath(int ArrayIndex)

{ // method to handle the death of this unit... put a 'dead picture' in its Symbol field

MyArrayOfMeleeUnits[ArrayIndex].Symbol = RTS\_Game\_Project\_18000551\_Jesse\_Hiebner.Properties.Resources.DeadMeleeBravo; //THIS NEEDS FIXING!!!!!

if (SoundHasAlreadyPlayed == false)

{

MySoundPlayer.Stream = RTS\_Game\_Project\_18000551\_Jesse\_Hiebner.Properties.Resources.DeathSound;

MySoundPlayer.Play();

SoundHasAlreadyPlayed = true;

}

} //end of MyDeath

public override string ToString()

{ // method to return a neatly formatted string showing all the unit?s information.

string CombatComment = " ";

if (IsAttacking == true) { CombatComment = "Yes!"; }

return "Melee: (" + X.ToString("00") + "," + Y.ToString("00") + ") " + Health.ToString("000") + "/" + MaxHealth.ToString("000") + " " + Speed.ToString("0") + " " + Attack.ToString("00") + " " + AttackRange.ToString("0") + " " + Faction + " " + String.Format("{0,5}",CombatComment) + " " + Name;

}

// Question 2.11 SAVE…. A string method to build a string of this unit’s info…which can be written to a csv file

public override string SaveSettingsAsString()

{ // converts the unit’s data to a string

string x, SymbolAsString;

x = Name + "," + X.ToString() + "," + Y.ToString() + "," + Health.ToString() + "," + MaxHealth.ToString() + "," + Speed.ToString() + "," + Attack.ToString() + "," + AttackRange.ToString() + "," + Faction;

SymbolAsString = ConvertImageToString(Symbol);

x = x + "," + SymbolAsString;

return x;

} // end of Save Settings

} // end of Melee unit

//Beginning of Ranged unit ---------------------------------------------------------------------------------------------------------------------------------------------

public class RangedUnit : Unit

{

// Question 1.6 Accessors

private int accessor\_X;

public new int X { get { return accessor\_X; } set { accessor\_X = value; } }

private int accessor\_Y;

public new int Y { get { return accessor\_Y; } set { accessor\_Y = value; } }

private int accessor\_Health;

public new int Health { get { return accessor\_Health; } set { accessor\_Health = value; } }

// private int accessor\_MaxHealth;

public new int MaxHealth { get; } //return accessor\_MaxHealth; } } //only Get

private int accessor\_Speed;

public new int Speed { get { return accessor\_Speed; } set { accessor\_Speed = value; } }

private int accessor\_Attack;

public new int Attack { get { return accessor\_Attack; } set { accessor\_Attack = value; } }

private int accessor\_AttackRange;

public new int AttackRange { get { return accessor\_AttackRange; } set { accessor\_AttackRange = value; } }

private string accessor\_Faction;

public new string Faction { get { return accessor\_Faction; } set { accessor\_Faction = value; } }

private Image accessor\_Symbol;

public new Image Symbol { get { return accessor\_Symbol; } set { accessor\_Symbol = value; } }

private bool accessor\_IsAttacking;

public new bool IsAttacking { get { return accessor\_IsAttacking; } set { accessor\_IsAttacking = value; } }

private string accessor\_Name;

public new string Name { get { return accessor\_Name; } set { accessor\_Name = value; } }

// constructor for Ranged .... which calls the parent UNIT constructor

public RangedUnit(int X, int Y, int Health, int Speed, int Attack, int AttackRange, string Faction, Image Symbol, bool IsAttacking)

: base(X, Y, Health, Speed, Attack, AttackRange, Faction, Symbol, IsAttacking)

{

// Question 1.5b ----- initialize Ranger with relevant values: -----

this.X = randomNumberGenerator.Next(2, 19); // initial column postion between 2 and 19

this.Y = randomNumberGenerator.Next(2, 19); // initial row postion between 2 and 19

this.Health = 50; // initial Health (physical strength) of RangedUnit is a lot less than a Melee

this.MaxHealth = this.Health; // Max health is set to initial health

this.Speed = 2; // this is actually the 'slowness' of the RangedUnit ? who moves at half the speed (every 2nd round)

this.Attack = 10; // a RangedUnit (eg bullets/arrows) deals double the attack-damage of a Melee

this.AttackRange = 3; // the ranged unit can attack (fire the gun) from up to 4 cell blocks away

int r = randomNumberGenerator.Next(1, 3); //Placing the unit in team Alpha or Bravo

if (r == 1)

{

this.Faction = "Alpha";

this.Symbol = RTS\_Game\_Project\_18000551\_Jesse\_Hiebner.Properties.Resources.RangedAlpha;

}

else

{

this.Faction = "Bravo";

this.Symbol = RTS\_Game\_Project\_18000551\_Jesse\_Hiebner.Properties.Resources.RangedBravo; // Bravo pictures have a black border

}

this.IsAttacking = false; // intitalize "is attacking" to false

// Question 2 - add Name property

r = randomNumberGenerator.Next(1, 4); //giving RangedUnits one of 3 random 'ranged' names

if (r == 1) { this.Name = "SniperKill"; }

if (r == 2) { this.Name = "RifleMan"; }

if (r == 3) { this.Name = "ShooterBoy"; }

}

//Overriden Methods: Question 1.4

public override void MoveToNewPosition(int Move)

{// this method sets the new X,Y position of a unit depending on what Move direction it goes to

// ie: 1=right, 2=left, 3=up, 4=down

if (Move == moveRight)

{

X = X + 1;

if (X > 19) { X = 19; }

}

if (Move == moveLeft)

{

X = X - 1;

if (X < 0) { X = 0; }

}

if (Move == moveUp)

{

Y = Y - 1;

if (Y < 0) { Y = 0; }

}

if (Move == moveDown)

{

Y = Y + 1;

if (Y > 19) { Y = 19; }

}

}// end to MoveToNewPosition

public override void HandleCombatWithEnemy(string EnemyUnitType, int EnemyArrayIndex)

{// method to attack an enemy... decrease his health with my attack-damge points

// set the IsAttacking field

IsAttacking = true;

if (EnemyUnitType == "Melee")

{

MyArrayOfMeleeUnits[EnemyArrayIndex].Health = MyArrayOfMeleeUnits[EnemyArrayIndex].Health - Attack;

if (MyArrayOfMeleeUnits[EnemyArrayIndex].Health < 0) { MyArrayOfMeleeUnits[EnemyArrayIndex].Health = 0; }

}

if (EnemyUnitType == "Ranged")

{

MyArrayOfRangedUnits[EnemyArrayIndex].Health = MyArrayOfRangedUnits[EnemyArrayIndex].Health - Attack;

if (MyArrayOfRangedUnits[EnemyArrayIndex].Health < 0) { MyArrayOfRangedUnits[EnemyArrayIndex].Health = 0; }

}

if (EnemyUnitType == "Resource")

{

MyArrayOfResourceBuildings[EnemyArrayIndex].Health = MyArrayOfResourceBuildings[EnemyArrayIndex].Health - Attack;

//Negative damage exception

if (MyArrayOfResourceBuildings[EnemyArrayIndex].Health < 0) { MyArrayOfResourceBuildings[EnemyArrayIndex].Health = 0; }

}

if (EnemyUnitType == "Factory")

{

MyArrayOfFactoryBuildings[EnemyArrayIndex].Health = MyArrayOfFactoryBuildings[EnemyArrayIndex].Health - Attack;

//Negative damage exception

if (MyArrayOfFactoryBuildings[EnemyArrayIndex].Health < 0) { MyArrayOfFactoryBuildings[EnemyArrayIndex].Health = 0; }

}

}

public override bool WithinRangeOfEnemy(string EnemyType, MeleeUnit MeleeEnemy, RangedUnit RangedEnemy)

{ // method to determine whether another unit is within attack range? returns true or false

bool withinRange = false;

if (EnemyType == "Melee")

{

if ((Math.Abs(X - MeleeEnemy.X) <= AttackRange) && (Math.Abs(Y - MeleeEnemy.Y) <= AttackRange))

{ withinRange = true; }

}

if (EnemyType == "Ranged")

{

if ((Math.Abs(X - RangedEnemy.X) <= AttackRange) && (Math.Abs(Y - RangedEnemy.Y) <= AttackRange))

{ withinRange = true; }

}

return withinRange;

} //within range of enemy

public override void PositionOfNearestEnemyUnitOrBuilding(ref int PosX, ref int PosY, ref string EnemyUnitType, ref int ArrayIndex)

{ // method to return position of the closest living enemy unit to me - via reference output parameters

// eg: output parameters will be: 12, 4, "Melee", 2 And this means .... the 2nd Melee unit in the array...whose X,y position is 12,4

int NearestDistance = 100;

int ThisDistance;

string Enemy;

if (Faction == "Alpha") { Enemy = "Bravo"; } else { Enemy = "Alpha"; }

//loop thru the Melee units looking for the nearest enemy

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

{

if ((MyArrayOfMeleeUnits[i].Faction == Enemy) && (MyArrayOfMeleeUnits[i].Health > 0))

{

ThisDistance = Math.Abs(X - MyArrayOfMeleeUnits[i].X) + Math.Abs(Y - MyArrayOfMeleeUnits[i].Y);

if (ThisDistance < NearestDistance)

{

NearestDistance = ThisDistance;

PosX = MyArrayOfMeleeUnits[i].X;

PosY = MyArrayOfMeleeUnits[i].Y;

EnemyUnitType = "Melee";

ArrayIndex = i;

}

}

}

//loop thru the Ranged units looking for an even nearer enemy

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

{

if ((MyArrayOfRangedUnits[i].Faction == Enemy) && (MyArrayOfRangedUnits[i].Health > 0))

{

ThisDistance = Math.Abs(X - MyArrayOfRangedUnits[i].X) + Math.Abs(Y - MyArrayOfRangedUnits[i].Y);

if (ThisDistance < NearestDistance)

{

NearestDistance = ThisDistance;

PosX = MyArrayOfRangedUnits[i].X;

PosY = MyArrayOfRangedUnits[i].Y;

EnemyUnitType = "Ranged";

ArrayIndex = i;

}

}

}

return;

} // end of Position of Nearest Enemy Unit

public override void MyDeath(int ArrayIndex)

{ // method to handle the death of this unit ... put a 'dead picture' in the Symbol property

MyArrayOfRangedUnits[ArrayIndex].Symbol = RTS\_Game\_Project\_18000551\_Jesse\_Hiebner.Properties.Resources.DeadRangedBravo; //THIS NEEDS FIXING!!!!!

if (SoundHasAlreadyPlayed == false)

{

MySoundPlayer.Stream = RTS\_Game\_Project\_18000551\_Jesse\_Hiebner.Properties.Resources.DeathSound;

MySoundPlayer.Play();

SoundHasAlreadyPlayed = true;

}

}

public override string ToString()

{ // method to return a neatly formatted string showing all the unit?s information.

string CombatComment = " ";

if (IsAttacking == true) { CombatComment = "Yes!"; }

return "Ranged: (" + X.ToString("00") + "," + Y.ToString("00") + ") " + Health.ToString("000")

+ "/" + MaxHealth.ToString("000") + " " + Speed.ToString("0") + " " + Attack.ToString("00")

+ " " + AttackRange.ToString("0") + " " + Faction + " " + String.Format("{0,5}", CombatComment) + " " + Name;

}

// Question 2.11 SAVE…. A string method to build a string of this unit’s info…which can be written to a csv file

public override string SaveSettingsAsString()

{ // converts the unit’s data to a string

string x, SymbolAsString;

x = Name + "," + X.ToString() + "," + Y.ToString() + "," + Health.ToString() + "," + MaxHealth.ToString() + "," + Speed.ToString() + ","

+ Attack.ToString() + "," + AttackRange.ToString() + "," + Faction;

SymbolAsString = ConvertImageToString(Symbol);

x = x + "," + SymbolAsString;

return x;

} // end of Save Settings

} // end of Ranged unit class

//\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

// Question 2.4 Create abstract base class called “Building”

public abstract class Building

{

protected int X;

protected int Y;

protected int Health;

protected int MaxHealth;

protected string Faction;

protected Image Symbol;

//-----Constructor for parent base Building ----------

public Building(int paramX, int paramY, int paramHealth, string paramFaction, Image paramSymbol)

{

X = paramX;

Y = paramY;

Health = paramHealth;

Faction = paramFaction;

Symbol = paramSymbol;

}

// abstract methods which will be overridden:

public abstract void MyDestruction(int ArrayIndex);

public abstract override string ToString();

//Question 2.11 ---- SAVE abstract method which will be overridden:

public abstract string SaveSettingsAsString();

} // end of Building base class --------------------------------------------------------------

//Question 2.5 Create child “ResourceBuilding” ---------------------------------------------------------------------------------------------

public class ResourceBuilding : Building

{

// new child variables

public string ResourceType; // a description eg 'Medical Supplies”'

public int ResourcesGenerated; // eg 60... a running total

public int ResourcesGeneratedPerRound; // eg 3 Medical Supplies per round

public int ResourcePoolRemaining; // eg 45 medical supplies remaining in the pool… stop generating resources if the pool is 0

// accessors for 'parent' variables

private int accessor\_X;

public new int X { get { return accessor\_X; } set { accessor\_X = value; } } //only needs GET ….as the building does not move…so the co-ordinates does not get re-set

private int accessor\_Y;

public new int Y { get { return accessor\_Y; } set { accessor\_Y = value; } } //only needs GET ….as the building does not move…so the co-ordinates does not get re-set

private int accessor\_Health;

public new int Health { get { return accessor\_Health; } set { accessor\_Health = value; } }

public new int MaxHealth { get; } //only needs GET

private string accessor\_Faction;

public new string Faction { get { return accessor\_Faction; } set { accessor\_Faction = value; } }

private Image accessor\_Symbol;

public new Image Symbol { get { return accessor\_Symbol; } set { accessor\_Symbol = value; } }

//constructor for Resource Building

public ResourceBuilding(int X, int Y, int Health, string Faction, Image Symbol) : base(X, Y, Health, Faction, Symbol)

{

this.X = randomNumberGenerator.Next(1, 20); // initial column position between 1..19

this.Y = randomNumberGenerator.Next(1, 20); // initial row position between 1..19

this.Health = 500; // initial Health (physical strength) of this building

this.MaxHealth = this.Health; // Max health is set to initial health

int r = randomNumberGenerator.Next(1, 3); //Placing the resource building in team Alpha or Bravo

if (r == 1)

{

this.Faction = "Alpha";

this.Symbol = RTS\_Game\_Project\_18000551\_Jesse\_Hiebner.Properties.Resources.ResourceBuildingAlpha;

}

else

{

this.Faction = "Bravo";

this.Symbol = RTS\_Game\_Project\_18000551\_Jesse\_Hiebner.Properties.Resources.ResourceBuildingBravo; // Bravo pictures have a black border

}

this.ResourceType = "Supplies";

this.ResourcesGenerated = 0; //running total of supplies generated during the game…starts off at 0.

this.ResourcesGeneratedPerRound = 5; //e.g. 5 medical supplies must be generated (i.e moved from the ‘pool’ to ‘ResourceGenerated’)

this.ResourcePoolRemaining = 300; // starting pool amount of 'medical' resources

} // end of Resource Building constructor

public void GenerateResourcesFromThePool()

{ // this method 'generates resources'… i.e it moves a fixed number of resources every round from the pool to the ResourcesGenerated counter

if (this.ResourcePoolRemaining >= this.ResourcesGeneratedPerRound)

{

this.ResourcesGenerated = this.ResourcesGenerated + this.ResourcesGeneratedPerRound;

this.ResourcePoolRemaining = this.ResourcePoolRemaining - this.ResourcesGeneratedPerRound;

}

else //move the last little bit out (e.g. 2) of the pool to the ResourcesGenerated counter…pool is now empty

{

this.ResourcesGenerated = this.ResourcesGenerated + this.ResourcePoolRemaining;

this.ResourcePoolRemaining = 0;

}

} //end of Generate Resources method

// over-ridden methods for the Resource Buildings

public override void MyDestruction(int ArrayIndex)

{ // method to handle the total destruction of this building ... put a black block in the Symbol property…also empty out is pool

MyArrayOfResourceBuildings[ArrayIndex].Symbol = RTS\_Game\_Project\_18000551\_Jesse\_Hiebner.Properties.Resources.DestroyedBuilding; //need a pic of black block!

this.ResourcePoolRemaining = 0;

if (SoundHasAlreadyPlayed == false)

{

MySoundPlayer.Stream = RTS\_Game\_Project\_18000551\_Jesse\_Hiebner.Properties.Resources.BuildingDestruction;

MySoundPlayer.Play();

SoundHasAlreadyPlayed = true;

}

}

public override string ToString()

{ // method to return a formatted string showing all the building's information.

return "Resource:" + "(" + X.ToString("00") + "," + Y.ToString("00") + ") " + Health.ToString("000") + "/" + MaxHealth.ToString("000") + " "

+ ResourcesGeneratedPerRound.ToString("00") + " " + ResourcesGenerated.ToString("000") + " "

+ ResourcePoolRemaining.ToString("000") + " " + String.Format("{0,-10}", Faction) + " " + String.Format("{0,-10}", ResourceType);

}

// Question 2.11 SAVE…. A string method to build a string of this BUILDING’s info…which can be written to a csv file

public override string SaveSettingsAsString()

{ // converts the unit’s data to a string

string x, SymbolAsString;

x = ResourceType + "," + X.ToString() + "," + Y.ToString() + "," + Health.ToString() + "," + MaxHealth.ToString() + ","

+ Faction + "," + ResourcesGenerated.ToString() + "," + ResourcesGeneratedPerRound.ToString() + "," + ResourcePoolRemaining.ToString();

SymbolAsString = ConvertImageToString(Symbol);

x = x + "," + SymbolAsString;

return x;

} // end of Save Settings

} // ------------ end of Resource Building child class ------------------------------------------------------

// Question 2.7 -------------------------- FACTORY BUILDING -------------------------------------------------------

public class FactoryBuilding : Building

{

// new child variables

public string UnitType; // eg Melee… this factory makes Melee soldiers

public int ProductionSpeed; // eg 10 …. Takes 10 rounds/seconds to make a new unit

public string SpawnPoint; // eg 'below' or 'above'

// accessors for 'parent' variables

private int accessor\_X;

public new int X { get { return accessor\_X; } set { accessor\_X = value; } } //only needs GET

private int accessor\_Y;

public new int Y { get { return accessor\_Y; } set { accessor\_Y = value; } } //only needs GET

private int accessor\_Health;

public new int Health { get { return accessor\_Health; } set { accessor\_Health = value; } }

public new int MaxHealth { get; } //only needs GET

private string accessor\_Faction;

public new string Faction { get { return accessor\_Faction; } set { accessor\_Faction = value; } } //only needs GET

private Image accessor\_Symbol;

public new Image Symbol { get { return accessor\_Symbol; } set { accessor\_Symbol = value; } }

//constructor for Factory Building

public FactoryBuilding(int X, int Y, int Health, string Faction, Image Symbol) : base(X, Y, Health, Faction, Symbol)

{

this.X = randomNumberGenerator.Next(1, 20); // initial column postion between 1 and 19

this.Y = randomNumberGenerator.Next(1, 20); // initial row postion between 1 and 19

this.Health = 500; // initial Health (physical strength) of Factory building

this.MaxHealth = this.Health; // Max health is set to initial health

int r = randomNumberGenerator.Next(1, 3); //Placing the Factory building in team Alpha or Bravo

if (r == 1)

{

this.Faction = "Alpha";

this.Symbol = RTS\_Game\_Project\_18000551\_Jesse\_Hiebner.Properties.Resources.FactoryBuildingAlpha;

}

else

{

this.Faction = "Bravo";

this.Symbol = RTS\_Game\_Project\_18000551\_Jesse\_Hiebner.Properties.Resources.FactoryBuildingBravo; // Bravo pictures have a black border

}

int FactoryRand = randomNumberGenerator.Next(1, 3); // does this factory make Melees or Rangeds ?

if (FactoryRand == 1) { this.UnitType = "Melee"; } else { this.UnitType = "Ranged"; }

this.ProductionSpeed = 20; // can only spawn a new unit every 20 rounds/seconds

this.SpawnPoint = "below"; //set spawn point

if (this.Y == 19) { this.SpawnPoint = "above"; }

} // end of Factory Building constructor

// Question 2.8 Create a method that spawns a unit

public void SpawnAnotherUnit()

{ // this method spawns another unit. The unit-type, faction, spawnpoint is determined by the factory's itself

if (this.UnitType.Equals("Ranged"))

{// instantiate a new Melee spawned unit

MeleeUnit U = new MeleeUnit(0, 0, 0, 0, 0, 0, "", null, false); // instantiates a new MeleeUnit (passing dummy parameter data to the parent class )

U.X = this.X; // the unit’s X position is the position of the factory

if (this.SpawnPoint.Equals("below")) { U.Y = this.Y + 1; } // the unit’s Y position is mostly 1 below the position of the factory ie Y+1

if (this.SpawnPoint.Equals("above")) { U.Y = this.Y-1; } // the unit’s Y position is above the factory ie Y-1

\_ = this.Faction; //READ ONLY. CANNOT BE ASSIGNED TO. Used to be U.Faction = this.Faction

// resize the array…. increase by 1

NumberOfMeleeUnits = NumberOfMeleeUnits + 1; // increase the number –of-melee counter by 1

Array.Resize(ref MyArrayOfMeleeUnits, NumberOfMeleeUnits);

MyArrayOfMeleeUnits[NumberOfMeleeUnits - 1] = U; // store the new unit at the end of the array

}

if (this.UnitType.Equals("Ranged"))

{ // instantiate a new Ranged spawned unit

RangedUnit U = new RangedUnit(0, 0, 0, 0, 0, 0, "", null, false); // instantiates a new Ranged Unit (passing dummy parameter data to the parent class )

U.X = this.X; // the unit’s X position is the position of this factory

if (this.SpawnPoint.Equals("below")) { U.Y = this.Y + 1; } // the unit’s Y position is mostly 1 below the position of this factory ie Y+1

if (this.SpawnPoint.Equals("above")) { U.Y = this.Y - 1; } // the unit’s Y position is above this factory ie Y-1

\_ = this.Faction; //READ ONLY. CANNOT BE ASSIGNED TO. Used to be U.Faction = this.Faction

// resize the array…. increase by 1

NumberOfRangedUnits = NumberOfRangedUnits + 1; // increase the number –of-ranged counter by 1

Array.Resize(ref MyArrayOfRangedUnits, NumberOfRangedUnits);

MyArrayOfRangedUnits[NumberOfRangedUnits - 1] = U; // store the new unit at the end of the array

}

} // end of SPAWN method

// over-ridden methods for the Factory Buildings

public override void MyDestruction(int ArrayIndex)

{ // method to handle the total destruction of this building ... put a black block in the Symbol property

MyArrayOfFactoryBuildings[ArrayIndex].Symbol = RTS\_Game\_Project\_18000551\_Jesse\_Hiebner.Properties.Resources.DestroyedBuilding;

if (SoundHasAlreadyPlayed == false)

{

MySoundPlayer.Stream = RTS\_Game\_Project\_18000551\_Jesse\_Hiebner.Properties.Resources.GlassSmash;

MySoundPlayer.Play();

SoundHasAlreadyPlayed = true;

}

}

public override string ToString()

{ // method to return a formatted string showing all the building's information.

return "Factory:" + " (" + X.ToString("00") + "," + Y.ToString("00") + ") " + Health.ToString("000") + "/"

+ MaxHealth.ToString("000") + " " + ProductionSpeed.ToString("00") + " "

+ SpawnPoint.ToString() + " " + String.Format("{0,-10}", UnitType) + " " + String.Format("{0,-10}", Faction);

}

public override string SaveSettingsAsString()

{ // converts the unit’s data to a string

string x, SymbolAsString;

x = UnitType + "," + X.ToString() + "," + Y.ToString() + "," + Health.ToString() + "," + MaxHealth.ToString() + ","

+ Faction + "," + ProductionSpeed.ToString() + "," + SpawnPoint.ToString();

SymbolAsString = ConvertImageToString(Symbol);

x = x + "," + SymbolAsString;

return x;

} // end of Save Settings

// ------------ end of Factory Building child class ------------------------------------------------------

}

//-----Question 1.7------ MAP CLASS -----------------------------------------------------------

// Genrate and display all units on the map

public class Map

{ //-----Question 1.7b----- MAP's constructor that receives the number of units to create - and sets the number of melees and rangeds

// ----Question 2.10 ………. also receive the number of buildings to create (eg 5) – and set the number of ResourceBuildings vs FactoryBuildings, randomly, eg 2 & 3

public Map(int NumberOfUnitsToCreate, int NumberOfBuildingsToCreate)

{// MAP constructor that receives the random number of units to create... and decides randomly how many should be Melee and how many Ranged

NumberOfMeleeUnits = randomNumberGenerator.Next(1, NumberOfUnitsToCreate); //note: this is a global integer

NumberOfRangedUnits = NumberOfUnitsToCreate - NumberOfMeleeUnits; // note: you will always have at least 1 of either type

// decide randomly how many buildings should be ResourceBuildings vs FactoryBuildings

NumberOfResourceBuildings = randomNumberGenerator.Next(1, (NumberOfBuildingsToCreate + 1)); //note: this is a global integer

NumberOfFactoryBuildings = NumberOfBuildingsToCreate - NumberOfResourceBuildings; // note: you will always have at least 1 of either type

}// end of MAP constructor

//-----Question 1.7c----- A method to create/generate the units ...o be stored in arrays

internal static void GenerateUnits(ref MeleeUnit[] paramMyArrayOfMeleeUnits, ref RangedUnit[] paramMyArrayOfRangedUnits)

{// A method to generate units. Some Melee, some Ranged - passed in as parameters

//Store units in arrays for easy referencing later

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

{

MeleeUnit U = new MeleeUnit(0, 0, 0, 0, 0, 0, "", null, false); // instantiates a new MeleeUnit (passing dummy data to the parent?)

paramMyArrayOfMeleeUnits[i] = U; // and stores it in an array

}

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

{

RangedUnit U = new RangedUnit(0, 0, 0, 0, 0, 0, "", null, false); // instantiates a new MeleeUnit (passing dummy data to the parent?)

paramMyArrayOfRangedUnits[i] = U; // and stores it in an array

}

}//End of GenerateUnits

//-----Question 2.10 ----- A method to generate buildings ... stored in arrays

internal static void GenerateBuildings(ref ResourceBuilding[] paramMyArrayOfResourceBuildings, ref FactoryBuilding[] paramMyArrayOfFactoryBuildings)

{

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

{

ResourceBuilding R = new ResourceBuilding(0, 0, 0, "", null); // instantiates a new ResourceBuilding (passing dummy parameter data to the parent class )

paramMyArrayOfResourceBuildings[i] = R; // and stores it in an array

}

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

{

FactoryBuilding F = new FactoryBuilding(0, 0, 0, "", null); // instantiates a newFactoryBuilding (passing dummy parameter data to the parent class )

paramMyArrayOfFactoryBuildings[i] = F; // and stores it in an array

}

} // end of generate buildings

//-----Question 1.7d----- Display all units on the map, as well as their information in the textbox on the right

internal static void DisplayAllUnits(MeleeUnit[] paramMyArrayOfMeleeUnits, RangedUnit[] paramMyArrayOfRangedUnits, RichTextBox MyRTB)

{

// first clear the grid of all units from the previous round

for (int i = 0; i <= 19; i++)

{

for (int j = 0; j <= 19; j++)

{ MyGrid.Rows[i].Cells[j].Value = null; }

}

// now clear the richtextbox

MyRTB.Text = null;

// declare empty units for temp storage

MeleeUnit M;

RangedUnit R;

//loop thru the Melee units

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

{

M = paramMyArrayOfMeleeUnits[i]; //extract the unit's info from the array

DisplaySpecificMeleeUnit(M); // and display its image on the grid

// also display the unit's info in the RichTextBox... using its ToString method

MyRTB.Text = MyRTB.Text + M.ToString() + Environment.NewLine;

}

//?similarly for Ranged units

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

{

R = paramMyArrayOfRangedUnits[i];

DisplaySpecificRangedUnit(R); //, MyGrid);

// also display the unit's info in the textbox... from its ToString method

MyRTB.Text = MyRTB.Text + R.ToString() + Environment.NewLine;

}

} //end of display all units

//-----Question 1.7e----- methods to display a specific unit on the map ... based on its changing X and Y coordinates

internal static void DisplaySpecificMeleeUnit(MeleeUnit SpecificMeleeUnit) //, MyTransparentDataGrid paramGrid)

{// display a Melee unit on the grid.... called by DisplayAllUnits

int x = SpecificMeleeUnit.X;

int y = SpecificMeleeUnit.Y;

Image Img = SpecificMeleeUnit.Symbol;

// use the 'attacking' symbol?... has a red border/star around it

if (SpecificMeleeUnit.IsAttacking && SpecificMeleeUnit.Faction == "Alpha") { Img = RTS\_Game\_Project\_18000551\_Jesse\_Hiebner.Properties.Resources.MeleeAlphaATTACKING; }

if (SpecificMeleeUnit.IsAttacking && SpecificMeleeUnit.Faction == "Bravo") { Img = RTS\_Game\_Project\_18000551\_Jesse\_Hiebner.Properties.Resources.MeleeBravoATTACKING; }

// display the image in the cells

MyGrid.Rows[y].Cells[x].Value = Img;

}

internal static void DisplaySpecificRangedUnit(RangedUnit SpecificRangedUnit) //, MyTransparentDataGrid paramGrid)

{// display one Ranged unit on the grid.... called by DisplayAllUnits

int x = SpecificRangedUnit.X;

int y = SpecificRangedUnit.Y;

Image Img = SpecificRangedUnit.Symbol;

// use the 'attacking' symbol?... has a red border/star around it

if (SpecificRangedUnit.IsAttacking && SpecificRangedUnit.Faction == "Alpha") { Img = RTS\_Game\_Project\_18000551\_Jesse\_Hiebner.Properties.Resources.RangedAlphaATTACKING; }

if (SpecificRangedUnit.IsAttacking && SpecificRangedUnit.Faction == "Bravo") { Img = RTS\_Game\_Project\_18000551\_Jesse\_Hiebner.Properties.Resources.RangedBravoATTACKING; }

// display the images in their cells

MyGrid.Rows[y].Cells[x].Value = Img;

}

// ----Question 2.10 --------- Display all buildings... as well as their information in the textbox on the right

internal static void DisplayAllBuildings(ResourceBuilding[] paramMyArrayOfResourceBuildings, FactoryBuilding[] paramMyArrayOfFactoryBuildings, RichTextBox MyRTB)

{

//Put a blank line in and add coloumn titles before displaying the resource buildings

MyRTB.Text = MyRTB.Text + Environment.NewLine;

//MyRTB.Text = MyRTB.Text + " Position Rate Counter Pool Team Description" + Environment.NewLine;

MyRTB.Text = MyRTB.Text + String.Format("{0,20}", "Position") + String.Format("{0,22}", "Rate") + String.Format("{0,9}", "Counter") + String.Format("{0,7}", "Pool")

+ String.Format("{0,8}", "Team") + String.Format("{0,21}", "Description") + Environment.NewLine;

// declare empty building variable for temp storage

ResourceBuilding R;

FactoryBuilding F;

//loop thru the ResourceBuildings

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

{

R = paramMyArrayOfResourceBuildings[i]; //extract the building’s info from the array

MyGrid.Rows[R.Y].Cells[R.X].Value = R.Symbol; // display the image in its cell

if(R.Health == 0)

{

MyGrid.Rows[R.Y].Cells[R.X].Value = RTS\_Game\_Project\_18000551\_Jesse\_Hiebner.Properties.Resources.DestroyedBuilding;

}

// also display the building’s info in the RichTextBox... using its ToString method

MyRTB.Text = MyRTB.Text + R.ToString() + Environment.NewLine;

}

//Put a blank line in and column titles before displaying the Factory buildings

MyRTB.Text = MyRTB.Text + "" + Environment.NewLine;

//“ Position Prod Spd SpawnPt Type Team”

MyRTB.Text = MyRTB.Text + String.Format("{0,20}", "Position") + String.Format("{0,23}", "Prod Spd") + String.Format("{0,15}", "SpawnPt")

+ String.Format("{0,10}", "Type") + String.Format("{0,17}", "Team") + Environment.NewLine;

//…similarly for FactoryBuildings

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

{

F = paramMyArrayOfFactoryBuildings[i];

MyGrid.Rows[F.Y].Cells[F.X].Value = F.Symbol; // display the image in its cell

if (F.Health == 0)

{ //Put in a black block if the buildings is destoryed

MyGrid.Rows[F.Y].Cells[F.X].Value = RTS\_Game\_Project\_18000551\_Jesse\_Hiebner.Properties.Resources.DestroyedBuilding;

}

// also display the building’s info in the RichTextBox... using its ToString method

MyRTB.Text = MyRTB.Text + F.ToString() + Environment.NewLine;

}

} //end of display all buildings

// Question 2 - Read the csv files and create units & buildings from it...then display them

internal static void ReadFilesAndDisplay(RichTextBox paramRTB)

{ // This method operates as follows:

// 1. Clear the grid, and screen

// 2. Read in Melee & Ranged unit files

// 3. Create arrays of Melee & Ranged units...and fill with dummy data

// 4. Loop thru Melee's textdata and write it to its array

// 5. loop thru Ranged textdata and write it to its array

// 6. Read the Resource & Factory Buildings csv file into its array

// 7. Create arrays of Resource & Factory builings...and fill with dummy data

// 8. Loop thru the resource building textline and populate the array

// 9. Read the FactoryBuildings csv file into its array

// 10. Display all units and buildings on the grid (call DisplayAllUnits and DisplayAllBuildings)

string FileName;

string AllFileContents;

// Step 1 Clear the grid, and screen

Application.OpenForms["MainForm"].Controls["lblWinner"].Text = null;

Application.OpenForms["MainForm"].Controls["RTB"].Text = null;

MyArrayOfMeleeUnits = null;

MyArrayOfRangedUnits = null;

MyArrayOfResourceBuildings = null;

MyArrayOfFactoryBuildings = null;

// Step 2 read in Melee & Ranged unit files

FileName = FileFolder + "\\GameSettingsMeleeUnits.CSV";

using (StreamReader MyStreamReader = new StreamReader(File.OpenRead(FileName)))

{

AllFileContents = MyStreamReader.ReadToEnd(); // read entire file into a string

}

// create array of text string lines - one line per unit - using the NewLine character as a split delimiter

string[] TextLinesMelee = AllFileContents.Split(Environment.NewLine.ToCharArray(), StringSplitOptions.RemoveEmptyEntries);

NumberOfMeleeUnits = TextLinesMelee.Length; // example 3

FileName = FileFolder + "\\GameSettingsRangedUnits.CSV";

using (StreamReader MyStreamReader = new StreamReader(File.OpenRead(FileName)))

{

AllFileContents = MyStreamReader.ReadToEnd(); // read entire file into a string

}

// create array of text string lines - one line per unit - using the NewLine character as a split delimiter

string[] TextLinesRanged = AllFileContents.Split(Environment.NewLine.ToCharArray(), StringSplitOptions.RemoveEmptyEntries);

NumberOfRangedUnits = TextLinesRanged.Length; // example 3

//Step 3 - Create arrays of Melee & Ranged units...and fill with dummy data

MeleeUnit[] localMyArrayOfMeleeUnits = new MeleeUnit[NumberOfMeleeUnits];

RangedUnit[] localMyArrayOfRangedUnits = new RangedUnit[NumberOfRangedUnits];

// generate the units randomly to fill the arrays with dummy data... NOTE: the data will be over-written)

GenerateUnits(ref localMyArrayOfMeleeUnits, ref localMyArrayOfRangedUnits);

// Step 4 - loop thru Melee's textdata and write it to its array

string[] TextFields;

//debug

string x; //debug

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

{

TextFields = TextLinesMelee[i].Split(','); // break up the line into 9 fields

// put the 10 fields into the array

localMyArrayOfMeleeUnits[i].Name = TextFields[0]; // Name

localMyArrayOfMeleeUnits[i].X = Int32.Parse(TextFields[1]); // x

localMyArrayOfMeleeUnits[i].Y = Int32.Parse(TextFields[2]); // x

localMyArrayOfMeleeUnits[i].Health = Int32.Parse(TextFields[3]); // Health

//localMyArrayOfMeleeUnits[i].MaxHealth = Int32.Parse(TextFields[4]); // MaxHealth is already set by the constructor!

localMyArrayOfMeleeUnits[i].Speed = Int32.Parse(TextFields[5]); // Health

localMyArrayOfMeleeUnits[i].Attack = Int32.Parse(TextFields[6]); // Attack

localMyArrayOfMeleeUnits[i].AttackRange = Int32.Parse(TextFields[7]); // Attack Range

localMyArrayOfMeleeUnits[i].Faction = TextFields[8]; // Faction

localMyArrayOfMeleeUnits[i].Symbol = ConvertStringToImage(TextFields[9].ToString());

}

// this now becomes the new array

MyArrayOfMeleeUnits = localMyArrayOfMeleeUnits;

// Step 5 - loop thru Ranged textdata and write it to its array

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

{

TextFields = TextLinesRanged[i].Split(','); // break up the line into 9 fields

// put the 9 fields into the array

localMyArrayOfRangedUnits[i].Name = TextFields[0]; // Name

localMyArrayOfRangedUnits[i].X = Int32.Parse(TextFields[1]); // X

localMyArrayOfRangedUnits[i].Y = Int32.Parse(TextFields[2]); // Y

localMyArrayOfRangedUnits[i].Health = Int32.Parse(TextFields[3]); // Health

//localMyArrayOfRangedUnits[i].MaxHealth = Int32.Parse(TextFields[4]); // MaxHealth is already set by the constructor!

localMyArrayOfRangedUnits[i].Speed = Int32.Parse(TextFields[5]); // Health

localMyArrayOfRangedUnits[i].Attack = Int32.Parse(TextFields[6]); // Attack

localMyArrayOfRangedUnits[i].AttackRange = Int32.Parse(TextFields[7]); // Attack Range

localMyArrayOfRangedUnits[i].Faction = TextFields[8]; // Faction

localMyArrayOfRangedUnits[i].Symbol = ConvertStringToImage(TextFields[9].ToString());

}

// this now becomes the new array

MyArrayOfRangedUnits = localMyArrayOfRangedUnits;

// step 6. Read the Resource & Factory Buildings csv file into its array

FileName = FileFolder + "\\GameSettingsResourceBuildings.CSV";

using (StreamReader MyStreamReader = new StreamReader(File.OpenRead(FileName)))

{

AllFileContents = MyStreamReader.ReadToEnd(); // read entire file into a string

}

// create array of text string lines - one line per unit - using the NewLine character as a split delimiter

string[] TextLinesResource = AllFileContents.Split(Environment.NewLine.ToCharArray(), StringSplitOptions.RemoveEmptyEntries);

NumberOfResourceBuildings = TextLinesResource.Length; // example 3

FileName = FileFolder + "\\GameSettingsFactoryBuildings.CSV";

using (StreamReader MyStreamReader = new StreamReader(File.OpenRead(FileName)))

{

AllFileContents = MyStreamReader.ReadToEnd(); // read entire file into a string

}

// create array of text string lines - one line per unit - using the NewLine character as a split delimiter

string[] TextLinesFactory = AllFileContents.Split(Environment.NewLine.ToCharArray(), StringSplitOptions.RemoveEmptyEntries);

NumberOfFactoryBuildings = TextLinesFactory.Length; // example 3

//Step 7 --- Create arrays of Resource & Factory builings...and fill with dummy data

ResourceBuilding[] localMyArrayOfResourceBuildings = new ResourceBuilding[NumberOfResourceBuildings];

FactoryBuilding[] localMyArrayOfFactoryBuildings = new FactoryBuilding[NumberOfFactoryBuildings];

// generate the buildings randomly to fill the arrays with dummy data... NOTE: the data will be over-written)

GenerateBuildings(ref localMyArrayOfResourceBuildings, ref localMyArrayOfFactoryBuildings);

//Step 8 --- Loop thru the resource building textline and populate the array

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

{

TextFields = TextLinesResource[i].Split(','); // break up the line into about 10 fields

// put the 10 fields into the array

localMyArrayOfResourceBuildings[i].ResourceType = TextFields[0]; // ResourceType

localMyArrayOfResourceBuildings[i].X = Int32.Parse(TextFields[1]); // X

localMyArrayOfResourceBuildings[i].Y = Int32.Parse(TextFields[2]); // Y

localMyArrayOfResourceBuildings[i].Health = Int32.Parse(TextFields[3]); // Health

//localMyArrayOfResourceBuildings[i].MaxHealth = Int32.Parse(TextFields[4]); // MaxHealth is already set by the constructor!

localMyArrayOfResourceBuildings[i].Faction = TextFields[5];

localMyArrayOfResourceBuildings[i].ResourcesGenerated = Int32.Parse(TextFields[6]); // Resources Generated

localMyArrayOfResourceBuildings[i].ResourcesGeneratedPerRound = Int32.Parse(TextFields[7]); // Resources Generated Per Round

localMyArrayOfResourceBuildings[i].ResourcePoolRemaining = Int32.Parse(TextFields[8]); // Resource Pool Remaining

localMyArrayOfResourceBuildings[i].Symbol = ConvertStringToImage(TextFields[9].ToString()); //Symbol

}

// this now becomes the new array

MyArrayOfResourceBuildings = localMyArrayOfResourceBuildings;

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

// step 9. Read the FactoryBuildings csv file into its array

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

{

TextFields = TextLinesFactory[i].Split(','); // break up the line into about 10 fields

// put the fields into the array

localMyArrayOfFactoryBuildings[i].UnitType = TextFields[0]; //UnitType

localMyArrayOfFactoryBuildings[i].X = Int32.Parse(TextFields[1]); // X

localMyArrayOfFactoryBuildings[i].Y = Int32.Parse(TextFields[2]); // Y

localMyArrayOfFactoryBuildings[i].Health = Int32.Parse(TextFields[3]); // Health

//localMyArrayOfFactoryBuildings[i].MaxHealth = Int32.Parse(TextFields[4]); // MaxHealth is already set by the constructor!

localMyArrayOfFactoryBuildings[i].Faction = TextFields[5]; // Faction

localMyArrayOfFactoryBuildings[i].ProductionSpeed = Int32.Parse(TextFields[6]); // ProductionSpeed

localMyArrayOfFactoryBuildings[i].SpawnPoint = TextFields[7]; // SpawnPoint

localMyArrayOfFactoryBuildings[i].Symbol = ConvertStringToImage(TextFields[8].ToString()); //Symbol WHICH IS NOT YET DONE

}

// this now becomes the new array

MyArrayOfFactoryBuildings = localMyArrayOfFactoryBuildings;

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

// step 10 Display all units and buildings on the grid (call DisplayAllUnits)

//Display units on the map, as well as populating the textbox report on the richtextbox 'RTB'….do the same for BUILDINGS!

MainForm.Map.DisplayAllUnits(localMyArrayOfMeleeUnits, localMyArrayOfRangedUnits, paramRTB); //Application.OpenForms["MainForm"].Controls["RTB"]);

MainForm.Map.DisplayAllBuildings(localMyArrayOfResourceBuildings, localMyArrayOfFactoryBuildings, paramRTB);

} // end of ReadFilesAndDisplay

}//End of Map Class

// ------GAME ENGINE --------------------------------------------------------------------------------

//-----Question 1.8----- Create a class called ?GameEngine?. --------------------------------------------------------

// This class will make changes to the game which will result in the changing of the ?Map?s? visual representation

public class GameEngine

{

// method to check if all Alpha units are dead

public static bool IsAllAlphaDead()

{// method to check if all Alpha units are dead

bool AllDead = true;

//loop thru the Melee units

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

{

if (MyArrayOfMeleeUnits[i].Faction == "Alpha")

{

if (MyArrayOfMeleeUnits[i].Health > 0) { AllDead = false; }

}

}

//loop thru the Ranged units

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

{

if (MyArrayOfRangedUnits[i].Faction == "Alpha")

{

if (MyArrayOfRangedUnits[i].Health > 0) { AllDead = false; }

}

}

return AllDead;

}

// method to check if all Bravo units are dead

public static bool IsAllBravoDead()

{// method to check if all Bravo units are dead

bool AllDead = true;

//loop thru the Melee units

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

{

if (MyArrayOfMeleeUnits[i].Faction == "Bravo")

{

if (MyArrayOfMeleeUnits[i].Health > 0) { AllDead = false; }

}

}

//loop thru the Ranged units

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

{

if (MyArrayOfRangedUnits[i].Faction == "Bravo")

{

if (MyArrayOfRangedUnits[i].Health > 0) { AllDead = false; }

}

}

return AllDead;

}

// a method to decide which direction a unit must advance toward an enemy? given my FROM position and the TO position of the enemy

public static int WhichDirectionToAdvance(int FromPosX, int FromPosY, int ToPosX, int ToPosY, int AttackRange)

{ // returns 1,2,3 or 4: 1-right, 2-left, 3-up, 4-down. Or 0 to stop if the distance is within my attackrange

// Note: FROM is me, the advancing unit ?. and TO is the enemy I must advance 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)

{

MoveHorizontally = true; //true means horizontal

}

else

{

MoveHorizontally = false; //false means vertical

}

if (MoveHorizontally == true) //Horizontal movement

{

if (FromPosX < ToPosX) { Direction = moveRight; } //Right

else { Direction = moveLeft; } //left

}

else //Vertical Movement

{

if (FromPosY > ToPosY) { Direction = moveUp; } //Up

else { Direction = moveDown; } //Down

}

// dont move ...if within attackrange!

if (HorizontalDif <= AttackRange && VerticalDif <= AttackRange)

{

Direction = 0;

}

return Direction;

} // end of WHICH DIRECTION TO ADVANCE

// a method to decide which direction a unit must retreat away from an enemy

// ? given my TO position and the FROM position of the enemy

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

{ // returns 1,2,3 or 4.... TO is ?me? wanting to retreat from the 'FROM' enemy

//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)

{

MoveHorizontally = false; //false means vertical

}

else

{

MoveHorizontally = true; //true means horizontal

}

if (MoveHorizontally == true) //Horizontal movement

{ // right or left

if (FromPosX <= ToPosX) { Direction = moveRight; } //Right

else { Direction = moveLeft; } //left

}

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

{

if (FromPosY >= ToPosY) { Direction = moveUp; } //Up

else { Direction = moveDown; } //Down

}

//Plan B ... what to do if you have hit the border while retreating

if (MoveHorizontally == false) // moving vertically towards border?

{

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

{

if (FromPosX >= ToPosX) { Direction = moveLeft; } // rather go left along the upper border

else { Direction = moveRight; } //rather go right along the upper border

}

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

{

if (FromPosX >= ToPosX) { Direction = moveLeft; } // rather go left along the bottom border

else { Direction = moveRight; } //rather go right along the bottom border

}

}

if (MoveHorizontally == true) // moving horizontally towards a border?

{

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

{

if (FromPosY >= ToPosY) { Direction = moveUp; } // rather go UP along the right border

else { Direction = moveDown; } //rather go DOWN along the right border

}

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

{

if (FromPosY >= ToPosY) { Direction = moveUp; } // rather go UP along the left border

else { Direction = moveDown; } //rather go DOWN along the left border

}

} // end of plan B...handling hitting the border

return Direction;

} // END OF FUNCTION - WHICH DIRECTION TO RETREAT

} // end of Game Engine class

}// end of Main Form

}// end end