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 sound efx!

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;

        //Random Number Generator:

        public static Random randomNumberGenerator = new Random();

        //-----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 = Properties.Resources.Map01\_ClanWilliam;  // I'm using an aerial picture of Clanwilliam (from Google Maps) as my battlefeld map

            }

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

            public Image MyBackgroundImage  // create a public property on the transparent grid... which uses MyPicture (Clanwilliam pic)

            {

                get { return MyPicture; }

                set { MyPicture = value; }

            }

            // override the PaintBackGround method of the dataviewgrid...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);  // draw MyBackgroundImage (property created above) over grid area

                // ... and also, make all the cells 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 called SettingPropertiesForMyGrid

        public void SettingPropertiesForMyGrid()

        {// ...add columns/rows to MyGrid:

            // adding 20 image columns to the empty MyGrid.... cos we want to display images in the 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;  //pics will be stretched to fit the block

                MyGrid.Columns.Add(imageColumn);   // and add the image column

            }

            //Now add 20 rows of the above columns:

            for (int i = 0; i < 20; i++) { MyGrid.Rows.Add(); }

            //set column widths and row heights to 25

            for (int i = 0; i < 20; i++) { MyGrid.Columns[i].Width = 25; }

            for (int i = 0; i < 20; i++) { MyGrid.Rows[i].Height = 25; }

            //set location of the grid map to point 20, 20 on the form

            MyGrid.Location = new System.Drawing.Point(20, 20);

            MyGrid.Size = new System.Drawing.Size(510, 510);    //set size of grid

            //Making the grid lines transparent.

            MyGrid.CellBorderStyle = DataGridViewCellBorderStyle.None;

            //Clear the selection/cursor ....

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

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

            MyGrid.ColumnHeadersHeight = 4;

            MyGrid.RowHeadersWidth = 4;

            // dont want scroll-bars on the grid

            MyGrid.ScrollBars = ScrollBars.None;

        }// end of setting properties for MyGrid - the map

        // load the MainForm and do initial settings

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

        {

            //Form size

            Width = 1100;

            Height = 600;

            //startup sounds !

            MySoundPlayer.Stream = Properties.Resources.Start\_of\_Play;

            MySoundPlayer.Play();

            // call SettingPropertiesForMyGrid to add rows/columns, etc to MyGrid:

            SettingPropertiesForMyGrid();

            // ...and now add the map (MyGrid) physically 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 EXIT\_Click(object sender, EventArgs e)

        {

            Close();

        }

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

        {

            //sound efx - explosion!

            MySoundPlayer.Stream = Properties.Resources.GunShot;

            MySoundPlayer.Play();

            // start the timer

            GameTimer.Start();

        }

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

        {

            GameTimer.Stop();

        }

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

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

            //sound efx - explosion

            MySoundPlayer.Stream = Properties.Resources.GunShot;

            MySoundPlayer.Play();

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

            lblWinner.Text = "";

            // creating a new 'map' .... instantiates a new Map...  randomly decide how many Melee vs Ranged (eg 4 & 6)

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

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

            //create 2 arrays of units...

            MeleeUnit[] localMyArrayOfMeleeUnits = new MeleeUnit[NumberOfMeleeUnits];

            RangedUnit[] localMyArrayOfRangedUnits = new RangedUnit[NumberOfRangedUnits];

            // generate the units randomly... and store them in the 2 arrays

            MyMap.GenerateUnits(ref localMyArrayOfMeleeUnits, ref localMyArrayOfRangedUnits);

            //Display units on the map, as well as populating the textbox report on the richtextbox 'RTB'

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

            //store the 2 arrays in global varaiables

            MyArrayOfMeleeUnits = localMyArrayOfMeleeUnits;

            MyArrayOfRangedUnits = localMyArrayOfRangedUnits;

        }// end of CreateClick

        // Question 2 - SAVE button

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

        {

            // write to file

        }

        // Question 2 - READ button

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

        {

            //reads from file

        }

        // GAME TIMER -----------------  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 & display a winning message & pause the game (a team wins if all the opponents are dead.. health<=0)

            //  Step 3 - Check if Melees can move in 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/retreat, then in what direction?

            //  Step 5 - Now repeat Step 3 & for Ranged units

            //  Step 7 -  Finally, display all units on the map, and their info in teh RichTextBox.... call MAP's DisplayAllUnits method...

            string MyTeam;

            int EnemyX, EnemyY, EnemyArrayIndex, Direction;

            string EnemyUnitType;

            //  Step 1 - display the round counter

            RoundNumber = RoundNumber + 1;

            lblRoundNumber.Text = RoundNumber.ToString();

            //  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 = Properties.Resources.CanonShot;

                MySoundPlayer.Play();

                lblWinner.Text = "ALPHA WINS!";

                GameTimer.Stop();

                return;  //exit

            }

            if (GameEngine.IsAllAlphaDead())

            {

                //sound efx - victory celebration

                MySoundPlayer.Stream = 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.....

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

                {

                    //reset the IsAttacking field back to false... in case it was in attack mode in the previous round

                    MyArrayOfMeleeUnits[i].IsAttacking = false;

                    //am I still alive?

                    if (MyArrayOfMeleeUnits[i].Health > 0)

                    {

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

                        EnemyX = 0;

                        EnemyY = 0;

                        EnemyArrayIndex = 0;

                        EnemyUnitType = "";

                        // calls method on the Melee unit

                        MyArrayOfMeleeUnits[i].PositionOfNearestEnemyUnit(ref EnemyX, ref EnemyY, ref EnemyUnitType, ref EnemyArrayIndex);

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

                        if (MyArrayOfMeleeUnits[i].Health < (0.25 \* MyArrayOfMeleeUnits[i].MaxHealth))

                        {   // retreat

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

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

                        }

                        else

                        {   // check if I am in attack-range... both X- and Y-distance must be within attack-range

                            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;

                                //sound efx

                                MySoundPlayer.Stream = Properties.Resources.GlassSmash;   // Melee's punch sounds liek a glass smash!

                                MySoundPlayer.Play();

                            }

                            else

                            {   // Advance toward enemy

                                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 MyDeath method... to replace the unit's 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

            {

                // Loop thru all Ranged units

                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)

                    {

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

                        EnemyX = 0;

                        EnemyY = 0;

                        EnemyArrayIndex = 0;

                        EnemyUnitType = "";

                        // calls method on the Ranged unit

                        MyArrayOfRangedUnits[i].PositionOfNearestEnemyUnit(ref EnemyX, ref EnemyY, ref EnemyUnitType, ref EnemyArrayIndex);

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

                        if (MyArrayOfRangedUnits[i].Health < (0.25 \* MyArrayOfRangedUnits[i].MaxHealth))

                        {

                            // Retreat

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

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

                        }

                        else

                        {   // Attack?

                            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;

                                //sound efx

                                MySoundPlayer.Stream = Properties.Resources.LaserShot;   // Ranged Unit has a laser gun !

                                MySoundPlayer.Play();

                            }

                            else

                            {

                                // Advance on the enemy

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

             Map.DisplayAllUnits(MyArrayOfMeleeUnits, MyArrayOfRangedUnits, 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 PositionOfNearestEnemyUnit(ref int PosX, ref int PoxY, ref string UnitType, ref int ArrayIndex);

            public abstract void MyDeath(int ArrayIndex);

            public abstract override string ToString();

        }  // end of UNIT class

        //Question 1.5 ----- Creating child classes

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

        public class MeleeUnit : Unit

        {

            // Question 1.6 Accessors..... this needs editing!!!!  the following does NOT need a SET accessor: Speed, Attack, Attack-Range, & Faction!!!

            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 needs a GET

            private int accessor\_Speed;

            public new int Speed { get { return accessor\_Speed; } set { accessor\_Speed = value; } }   // only needs a GET

            private int accessor\_Attack;

            public new int Attack { get { return accessor\_Attack; } set { accessor\_Attack = value; } }    // only needs a GET

            private int accessor\_AttackRange;

            public new int AttackRange { get { return accessor\_AttackRange; } set { accessor\_AttackRange = value; } }     // only needs a GET

            private string accessor\_Faction;

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

            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; } } //  only needs a GET

            // 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, string Name)

                                                                : 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 = Properties.Resources.MeleeAlpha;

                }

                else

                {

                    this.Faction = "Bravo";

                    this.Symbol = 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 my IsAttacking field

                this.IsAttacking = true;

                if (EnemyUnitType == "Melee")

                {

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

                }

                if (EnemyUnitType == "Ranged")

                {

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

                }

            } // 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(this.X - MeleeEnemy.X) <= AttackRange) && (Math.Abs(this.Y - MeleeEnemy.Y) <= this.AttackRange))

                    {

                        withinRange = true;

                    }

                }

                if (EnemyType == "Ranged")

                {

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

                    {

                        withinRange = true;

                    }

                }

                return withinRange;

            } // end of withinRangeOfEnemy

            public override void PositionOfNearestEnemyUnit(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(this.X - MyArrayOfMeleeUnits[i].X) + Math.Abs(this.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(this.X - MyArrayOfRangedUnits[i].X) + Math.Abs(this.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 its Symbol field

                MyArrayOfMeleeUnits[ArrayIndex].Symbol = Properties.Resources.DeadMelee;

            } //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 = "In Combat!"; }

                return "Melee: " + String.Format("{0,-10}", Name) + "(" + X.ToString("00") + "," + Y.ToString("00") + ")    " + Health.ToString("000") + "/" + MaxHealth.ToString("000") + " " + Speed.ToString("0") + " " + Attack.ToString("00") + "            " + AttackRange.ToString("0") + " " + Faction + " " + CombatComment;

            }

        } // end of Melee unit

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

        public class RangedUnit : Unit

        {

            // Question 1.6 Accessors.......this needs editing!!!!  the following does NOT need a SET accessor: Speed, Attack, Attack-Range, & Faction!!!

            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\_Health; } }  //only needs GET

            private int accessor\_Speed;

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

            private int accessor\_Attack;

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

            private int accessor\_AttackRange;

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

            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; } } //  only needs a GET

            // 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 half of a Melee's

                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's bullets deals double the attack-damage of a Melee's punch

                this.AttackRange = 4;                            // 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 = Properties.Resources.RangedAlpha;

                }

                else

                {

                    this.Faction = "Bravo";

                    this.Symbol = 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 - this.Attack;

                }

                if (EnemyUnitType == "Ranged")

                {

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

                }

            }

            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) <= this.AttackRange))

                           { withinRange = true; }

                }

                if (EnemyType == "Ranged")

                {

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

                            { withinRange = true; }

                }

                return withinRange;

            }  //within range of enemy

            public override void PositionOfNearestEnemyUnit(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(this.X - MyArrayOfMeleeUnits[i].X) + Math.Abs(this.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(this.X - MyArrayOfRangedUnits[i].X) + Math.Abs(this.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 = Properties.Resources.DeadRanged;

            }

            public override string ToString()

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

                string CombatComment = " ";

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

                return "Ranged: " + String.Format("{0,-10}", Name) + "(" + X.ToString("00") + "," + Y.ToString("00") + ")    " + Health.ToString("000") + "/" + MaxHealth.ToString("000") + " " + Speed.ToString("0") + " " + Attack.ToString("00") + "            " + AttackRange.ToString("0") + " " + Faction + " " + CombatComment;

            }

        } // end of Ranged unit class

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

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

        //  Generate 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 Ranged Units

            public Map(int NumberOfUnitsToCreate)

            {// 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

            }// end of MAP constructor

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

            public 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++)

                {   // instantiates a new MeleeUnit (passing dummy parameter data to the parent class )

                    MeleeUnit U = new MeleeUnit(0, 0, 0, 0, 0, 0, "", null, false);

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

                }

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

                {   // instantiates a new RangedUnit (passing dummy parameter data to the parent class )

                    RangedUnit U = new RangedUnit(0, 0, 0, 0, 0, 0, "", null, false);

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

                }

            }//End of GenerateUnits

            //-----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 pictures 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; }

                }

                // ...and 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 star on it

                if (SpecificMeleeUnit.IsAttacking && SpecificMeleeUnit.Faction == "Alpha") { Img = Properties.Resources.MeleeAlphaATTACKING; }

                if (SpecificMeleeUnit.IsAttacking && SpecificMeleeUnit.Faction == "Bravo") { Img = Properties.Resources.MeleeBravoTTACKING; }

                // 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 star on it

                if (SpecificRangedUnit.IsAttacking && SpecificRangedUnit.Faction == "Alpha") { Img = Properties.Resources.RangedAlphaATTACKING; }

                if (SpecificRangedUnit.IsAttacking && SpecificRangedUnit.Faction == "Bravo") { Img = Properties.Resources.RangedBravoATTACKING; }

                // display the images in their cells

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

            }

        }//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