Contents

[**1.** **UML CASE DIAGRAM AND CLASS DIAGRAM** 3](#_Toc533368269)

[**2.** **MY SYSTEM** 6](#_Toc533368270)

[**2.1. Strengths** 6](#_Toc533368271)

[**2.2. Weaknesses** 6](#_Toc533368272)

[**3.** **SYSTEM DESCRIPTION** 7](#_Toc533368273)

[**3.1. Source code** 7](#_Toc533368274)

[**2.2. How does it work?** 69](#_Toc533368275)

[**3.2. Screenshots** 69](#_Toc533368276)

[**3.4. Self-Assessment Sheet** 78](#_Toc533368277)

[**4.** **REFERENCES** 80](#_Toc533368278)

# **UML CASE DIAGRAM AND CLASS DIAGRAM**

I chose to do the third option (Spell to Score game) from the list of examples of games and quizzes that were given to us because I wanted to do something challenging and complex. I wanted to code something that would profoundly test my coding skills and that in the process, it would make me acquire more programming and development knowledge or at least it would strengthen my programming abilities and skills.

Another reason for choosing exactly this game was because I saw that it had the potential to meet the learning demands and criteria that were asked from it and it could be turned into a fun game at the same time. Even thought my idea of the game is little bit different from the one from the example the concept is still the exact same.

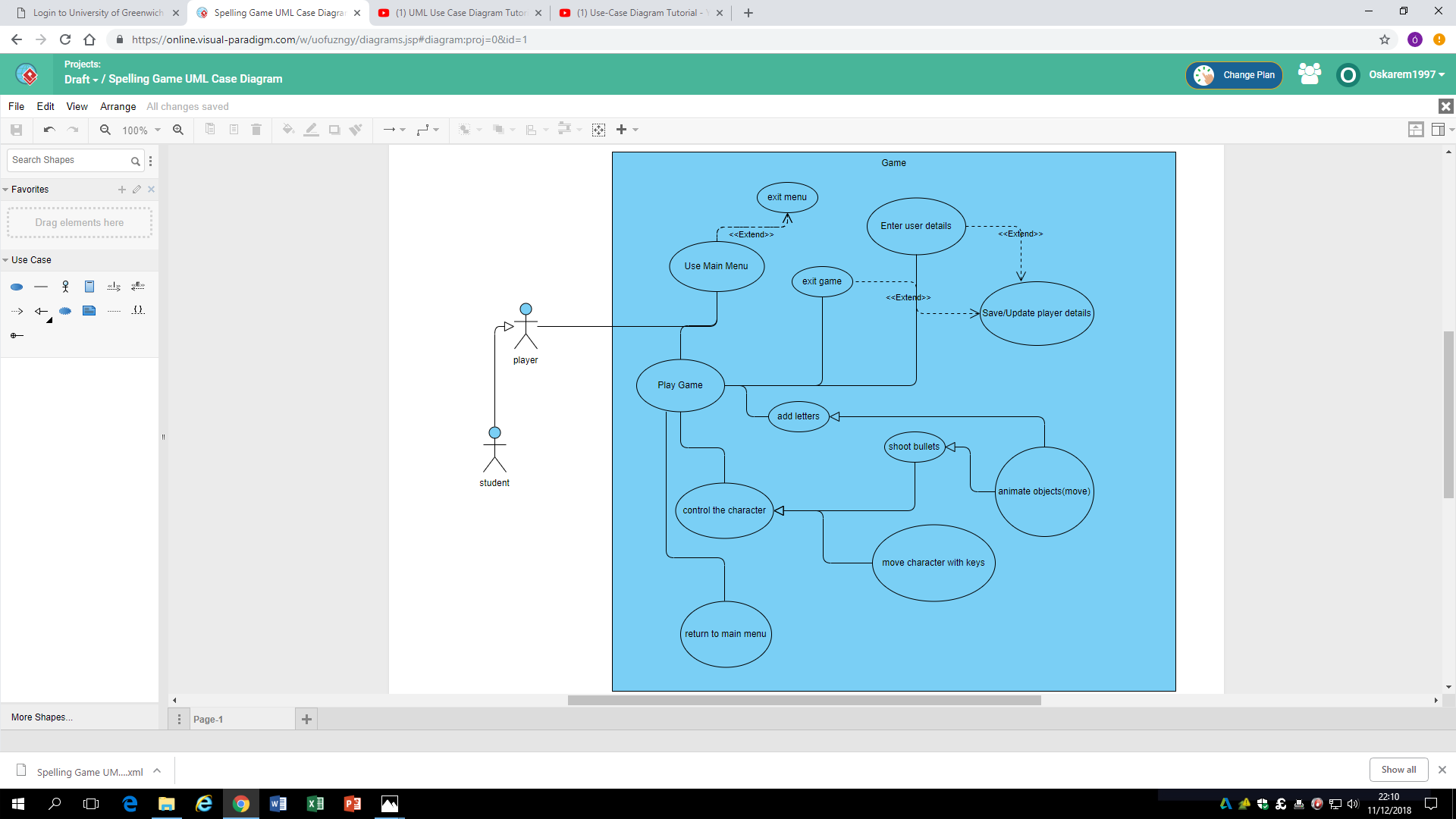
Planning the design of this game was not an easy task since I had to find a way to include most, if not all the material learned from the lectures, tutorials and books. In addition to this, the hackathon also inspired me into designing my application project like the Flying Pig program.

As can be seen from the UML case diagram the design of the application is simple. My idea was to create a game that resembled the different games that I have played in terms of functionality. Like a menu, user setting a nickname, etc.

I decided to implement the spelling game version because it was the option that was first specified. On the other hand, the idea of the square game version was very appealing to me, so I decided that if I still had time after finishing the spelling game, I could implement the other option as an extra.

As it is showed in the class diagram the game is formed by several classes, one interface and two interfaces. The more important classes are the player class, the letter class, the bullet class and the window where the game is being played.

Finally, with character on the screen, random letters (belonging to a word) will emerge on the screen and the character will have to shoot at all of them in certain or der to match the word being displayed in the screen.



A screen shot of a computer

Description automatically generated

# **MY SYSTEM**

## **2.1. Strengths**

The code in my application is well divided and structured. Each important aspect the game has its own class making it difficult to affect each other negatively. And in the case that there is a bug or any logic error; the bug or error is easily detected because you will know exactly in which class the mistake is being done depending on when or where the error has occurred. As well as this, the naming of all my variable fields, properties and methods are accurate and describe exactly what they were created for. This makes it easier for me or any other programmers to know what is going on in my program and makes it more effective to debug when looking for mistakes, if there are any.

As well as this, for the animation; I use three different timers to animate the objects in my game. This is very useful because I can animate different objects in my game at different speeds and get the results that I am really looking for. They also help with the logic in certain parts of the game, for example collision detection, and to check if the spellings are correct. My letter class also inherits from the Image class which allows me to use its functions and properties.

Using threads to make the animation has proven to be very effective and useful. Since most of the animations are carried by the primary thread (moving the player and the bullets), a second thread takes a load off the first one by overseeing animating the letters.

And finally, I have switched from windows form to WPF. In terms if graphics WPF comes with a lot more of advantages than windows forms. Besides from the better-looking layout of the windows, its easier to use different controls together in the same windows. I was not able to make my application look the way I wanted with windows form and because I already have some experience with WPF I decided to switch. What I wanted to achieve was to make an object rotate or to follow the direction of the motion of cursor which I was not able to do in windows forms.

## **2.2. Weaknesses**

The only weaknesses that my system has is the use of many global variables. I am using large quantities of global variables in my classes to keep track of animations, although majority are private. But because I am reusing them or calling them in plenty other several methods it creates bugs in my code.

There three bugs in my system that I am aware of and all of them are difficult to find because of the very difficult task of keeping track of my global variables. The bugs are the following.

First, keeping the character of the game within the limits of my canvas does not work properly.

Second, there are times in which a letter in the canvas disappears by it self without being hit by a bullet nor by crossing the limits of the canvas.

Third, if the cursor is put on the same place as the player, the program stops until the cursor is taken away

# **SYSTEM DESCRIPTION**

## **3.1. Source code**

namespace Application\_Dev\_Project

{

//Opens new windows

class windowOpener

{

private string newgame;

private string userNick;

private bool checkigUserNameWin;

public windowOpener(typeOfWindow screen, string s, string nickname)

{

userNick = nickname;

newgame = s;

if (screen == typeOfWindow.gameScreen)

{

OpenAnewPlayingScreen();

}

else if (screen == typeOfWindow.mainMenu)

{

}

else if (screen == typeOfWindow.settings)

{

}

else if (screen == typeOfWindow.instructions)

{

}

else if (screen == typeOfWindow.userInput)

{

OpenUserInput();

}

}

internal string getTypeOfGame()

{

return newgame;

}

private void OpenAnewPlayingScreen()

{

playGameScreen aNewScreen = new playGameScreen(getTypeOfGame(),userNick);

aNewScreen.Show();

}

private void OpenAnewMainMenu(object obj)

{

MainWindow aNewScreen = new MainWindow();

aNewScreen.Show();

}

private void OpenSettings()

{

}

private void OpenInstructions()

{

}

private void OpenUserInput()

{

UserName uNWindow = new UserName();

do

{

uNWindow.ShowDialog();

checkigUserNameWin = uNWindow.Continue();

} while (checkigUserNameWin==false);

}

}

}

namespace Application\_Dev\_Project

{

public enum typeOfWindow

{

mainMenu, settings, instructions, gameScreen,

userInput

}

}

namespace Application\_Dev\_Project

{

//Player clase made of a line and a an ellipse

public class Player : IGameEngine

{

public int X { get; set; }

public int Y { get; set; }

// public double xPlayer;

// public double yplayer;

double newx = 0;//new x2 coordinate of the fireArm

double newy = 0;//new y2 coordinate of the fireArm

public int Firearmlength { get; private set; }

public Point newPoint = new Point();

bool checkMouse = false;//checks if the mouse has entered the canvas

bool keypressed = false;//checks if a key has been pressed

public double temp1;//x1 coordinate of the rotating point

public double temp2;//y1 coordinate of the rotating point

public double temp3;//x2 coordinate of the rotating point

public double temp4;//y2 coordinate of the rotating point

double specialX = 0;

double specialY = 0;

Ellipse refreshedbody = new Ellipse();

Line refreshedFireArm = new Line();

bool checkBodyBoundaries = false;

bool checkFireArmBoundaries = false;

List<Line> locSaved = new List<Line>();

List<Point> locBodySaved = new List<Point>();

int velocity = 5;

int velocityMovementx;

int velocityMovementy;

Canvas playerCanv;

public Player(int x1, int y1, Canvas c)

{

playerCanv = c;

X = x1;

Y = y1;

refreshedbody = theBody();

refreshedFireArm = fireArm();

}

public void move(Directions move)

{

//Removing the old drawing from the canvas

playerCanv.Children.Remove(refreshedbody);

playerCanv.Children.Remove(refreshedFireArm);

keypressed = true;

//gets the last coordinates after every move

newPoint.X = newx;

newPoint.Y = newy;

if (move == Directions.up)

{

//updates the position travalled in the vertical sense

velocityMovementy = velocityMovementy - velocity;

specialY = specialY - 5;

}

else if (move == Directions.down)

{

//updates the position travalled in the vertical sense

velocityMovementy = velocityMovementy + velocity;

specialY = specialY + 5;

}

if (move == Directions.right)

{

//updates the position travalled in the horizontal sense

velocityMovementx = velocityMovementx + velocity;

specialX = specialX + 5;

}

else if (move == Directions.left)

{

//updates the position travalled in the horizontal sense

velocityMovementx = velocityMovementx - velocity;

specialX = specialX - 5;

}

//redrawing the body in the new location

refreshedbody = theBody();

refreshedFireArm = fireArm();

keypressed = false;

}

public Ellipse theBody()

{

playerCanv.Children.Remove(refreshedbody);

Ellipse body = new Ellipse();

TransformGroup bodyTransGroup = new TransformGroup();

TranslateTransform bodyTranslate = new TranslateTransform();

Point bodyLocation = new Point();

body.Width = 50;

body.Height = 50;

SolidColorBrush sb = new SolidColorBrush(Colors.Black);

body.Fill = sb;

bodyTranslate.X = X + bodyTranslate.X + velocityMovementx;

bodyTranslate.Y = Y + bodyTranslate.Y + velocityMovementy;

//Sets the body to stay inside the boundaries

bodyLocation.X = bodyLocation.X + bodyTranslate.X;

bodyLocation.Y = bodyLocation.Y + bodyTranslate.Y;

Point aTempPoint = new Point();

aTempPoint = bodyLocation;

int lastPointIndex = 0;

savebODYLocation(aTempPoint);

//Saves the locations of the player

if (bodyLocation.X <= 1)

{

lastPointIndex = locBodySaved.LastIndexOf(aTempPoint);

checkBodyBoundaries = true;

bodyTranslate.X = 1 ;

bodyLocation.X = 1;

}

else if ((bodyLocation.X >= playerCanv.ActualWidth - 50) && playerCanv.ActualWidth - 50 > 0)

{

lastPointIndex = locBodySaved.LastIndexOf(aTempPoint);

checkBodyBoundaries = true;

bodyTranslate.X = playerCanv.ActualWidth - 50 ;

bodyLocation.X = playerCanv.ActualWidth - 50;

}

if (bodyLocation.Y <= 1)

{

lastPointIndex = locBodySaved.LastIndexOf(aTempPoint);

checkBodyBoundaries = true;

bodyTranslate.Y = 1 /\*locBodySaved[locBodySaved.Count-3].X\*/;

bodyLocation.Y = 1;

//bodyTranslate.Y = locBodySaved[lastPointIndex - 1].Y;

//bodyLocation.Y = locBodySaved[lastPointIndex - 1].Y;

}

else if ((bodyLocation.Y >= playerCanv.ActualHeight - 50) && playerCanv.ActualHeight - 50 > 0)

{

savebODYLocation(aTempPoint);

checkBodyBoundaries = true;

bodyTranslate.Y = playerCanv.ActualHeight - 50/\*locBodySaved[locBodySaved.Count-3].X\*/;

bodyLocation.Y = playerCanv.ActualHeight - 50;

//bodyTranslate.Y = locBodySaved[lastPointIndex - 1].Y;

//bodyLocation.Y = locBodySaved[lastPointIndex - 1].Y;

}

bodyTransGroup.Children.Add(bodyTranslate);

body.RenderTransform = bodyTransGroup;

//player Redrawn

playerCanv.Children.Add(body);

return body;

}

//line made up of two poits

//2nd point is where the bullets "come out" from

public Line fireArm()

{

//Removes fire arm from previous locations

playerCanv.Children.Remove(refreshedFireArm);

Line fireArm = new Line();

TranslateTransform fireArmbodyTranslate = new TranslateTransform();

TransformGroup fireArmbodyTransGroup = new TransformGroup();

fireArm.Stroke = Brushes.Black;

fireArm.StrokeThickness = 10;

// double x1bound, x2bound, y1bound, y2bound;

//dynamicaly redraws every single new location

//gives the impression of moving and rotating

fireArm.X1 = fireArm.X1 + X + 25 + velocityMovementx;

fireArm.Y1 = fireArm.Y1 + Y + 25 + velocityMovementy;

temp1 = fireArm.X1;

temp2 = fireArm.Y1;

//if mouse outside the canvas control and player is being moved

if (checkMouse == false && keypressed == false)

{

fireArm.X2 = fireArm.X2 + X + 75 + velocityMovementx;

fireArm.Y2 = fireArm.Y2 + Y + 25 + velocityMovementy;

temp3 = fireArm.X2;

temp4 = fireArm.Y2;

velocityMovementx = 0;

velocityMovementy = 0;

}

if (checkMouse == true && keypressed == false)

{

fireArm.X2 = fireArm.X2 + newx;

fireArm.Y2 = fireArm.Y2 + newy;

temp3 = fireArm.X2;

temp4 = fireArm.Y2;

}

if (checkMouse == false && keypressed == true)

{

if (newPoint.X == 0 && newPoint.Y == 0)

{

fireArm.X2 = fireArm.X2 + X + 75 + velocityMovementx;

fireArm.Y2 = fireArm.Y2 + Y + 25 + velocityMovementy;

temp3 = fireArm.X2;

temp4 = fireArm.Y2;

}

else

{

fireArm.X2 = fireArm.X2 + newPoint.X + specialX;

fireArm.Y2 = fireArm.Y2 + newPoint.Y + specialY;

temp3 = fireArm.X2;

temp4 = fireArm.Y2;

}

}

//prevents the pregram from breaking

if (temp3 == 0)

{

temp3 = 0.1;

}

else if (temp4 == 0)

{

temp4 = 0.1;

}

//backup fireArm

Line tempLine = new Line();

tempLine = fireArm;

savePreviousLocation(tempLine);

//boundaries. "keeps" the player from going out of bounds

//it does not work properly. there is a bug in THIS class

if (fireArm.X2 <= 1 || fireArm.X1 <= 1)

{

checkFireArmBoundaries = true;

fireArm.X2 = locSaved[locSaved.Count - 2].X2;

fireArm.X1 = locSaved[locSaved.Count - 2].X1;

}

else if ((fireArm.X2 >= playerCanv.ActualWidth || fireArm.X1 >= playerCanv.ActualWidth) && playerCanv.ActualWidth > 0)

{

checkFireArmBoundaries = true;

fireArm.X2 = locSaved[locSaved.Count - 2].X2;

fireArm.X1 = locSaved[locSaved.Count - 2].X1;

}

if (fireArm.Y2 <= 1 || fireArm.Y1 <= 1)

{

checkFireArmBoundaries = true;

fireArm.Y2 = locSaved[locSaved.Count - 2].Y2;

fireArm.Y1 = locSaved[locSaved.Count - 2].Y1;

}

else if ((fireArm.Y2 >= playerCanv.ActualHeight || fireArm.Y1 >= playerCanv.ActualHeight) && playerCanv.ActualHeight > 0)

{

checkFireArmBoundaries = true;

fireArm.Y2 = locSaved[locSaved.Count - 2].Y2;

fireArm.Y1 = locSaved[locSaved.Count - 2].Y1;

}

fireArm.RenderTransformOrigin = new Point(fireArm.X1 / temp3, fireArm.Y1 / temp4);

fireArmbodyTransGroup.Children.Add(fireArmbodyTranslate);

fireArm.RenderTransform = fireArmbodyTransGroup;

playerCanv.Children.Add(fireArm);

return fireArm;

}

//Calculates the location of a new point every time the mouse is inside the canvas control to create the fireAre(fireArm is a line )

public void rotate(Point location, bool mouseinScreen)

{

playerCanv.Children.Remove(refreshedbody);

playerCanv.Children.Remove(refreshedFireArm);

checkMouse = mouseinScreen;

double distantanceMtoB;

distantanceMtoB = Math.Sqrt(Math.Pow((int)(location.X - temp1), 2) + Math.Pow((int)(location.Y - temp2), 2));

//Avoids sigularitites when the mouse is places in the same place as the first point of the fireArm

if (distantanceMtoB == 0)

{

distantanceMtoB = 1;

}

//calculates the new coordinates(x,y) of the line relative to the position of the mouse to redraw it

newx = ((50 / distantanceMtoB) \* (location.X - temp1)) + temp1;

newy = ((50 / distantanceMtoB) \* (location.Y - temp2)) + temp2;

refreshedbody = theBody();

refreshedFireArm = fireArm();

specialX = 0;//resets it to 0 to keep the firearm from losing its shape

specialY = 0;//resets it to 0 to keep the firearm from losing its shape

checkMouse = false;

}

void savePreviousLocation(Line a)

{

locSaved.Add(a);

}

void savebODYLocation(Point a)

{

locBodySaved.Add(a);

}

}

}

namespace Application\_Dev\_Project

{

class Number : System.Windows.Controls.Image, IGameEngine

{

public Number()

{

}

public void move(Directions move)

{

}

public void rotate(System.Windows.Point location, bool checkmouseInScreen)

{

// throw new NotImplementedException();

}

}

}

namespace Application\_Dev\_Project

{

abstract class movingObject

{

//First designed for polymorphism

public virtual void move(Directions move) { }

public virtual void rotate(Point location, bool checkmouseInScreen) { }

}

}

namespace Application\_Dev\_Project

{

/// <summary>

/// Interaction logic for MainWindow.xaml

/// </summary>

public partial class MainWindow : Window

{

public MainWindow()

{

InitializeComponent();

WindowStartupLocation = System.Windows.WindowStartupLocation.CenterScreen;

}

private void playGameBtn\_Click(object sender, RoutedEventArgs e)

{

//Open new windows

windowOpener newPlayingWindow = new windowOpener(typeOfWindow.userInput, "","");

this.Close();

}

private void settingsBtn\_Click(object sender, RoutedEventArgs e)

{

}

private void instructionsBtn\_Click(object sender, RoutedEventArgs e)

{

}

private void exitBtn\_Click(object sender, RoutedEventArgs e)

{

//closes the window when cliked

this.Close();

}

}

}

namespace Application\_Dev\_Project

{

/// <summary>

/// Interaction logic for playGameScreen.xaml

/// </summary>

public partial class playGameScreen : Window

{

bool reset;//checks if the textblock can be reset to "x" whenever a new word is spelled correctly

int letterOrder = 0;//counts how many letters have been hit by a bullet

Player thePlayer;

Point mousePosition { get; set; }

DispatcherTimer gameTimer;

DispatcherTimer bulletsTimer;//animates the bullets

DispatcherTimer lettersTimer;//animates the letters

List<bullets> theBullets;

List<letters> theLetters;

List<Number> theNumbers;

//Number aNumber;

//letters aLetter;

private int letterInterval;//interval between teh emergence of a letter and another one

private string theTypeOfGame;//checks what type of game is going to be played

private int bulletCount;//increments every time a bullets is fired

int theTime = 0;

int countWordsInterv = 0;//interval between teh emergence of a word and another one

public string getLetter="";//gets new lleter from list of letters

Thread theSpellingGame;//starts a thread to play the game

//Thread theSquaresGame;

List<String> listofWords;

//Image theimage;

//BitmapImage theBitImg;

int countToStartSpelling = 0;

int wordindex = 0;

int tempint = 0;

bool newWord=true;

int startshooting;

int thepoints;

int realSCore = 0;//checks the score

int indexLetter=0;

bool finishGame = false;

private bool GameOver=false;

bool initiatingLetters = false;

private int timeRemaining;

string theWord = "";

private string spelledWord;

private string userNickName;//gets the player nickname

public playGameScreen(string c, string nick)

{

InitializeComponent();

WindowStartupLocation= System.Windows.WindowStartupLocation.CenterScreen;

setTypeOfGame(c);

setUserNick(nick);

//string getLetter;

gameTimer = new DispatcherTimer();

gameTimer.Tick += gameTimer\_Tick;

gameTimer.Interval = new TimeSpan(0, 0, 0, 1, 0);

//theBitImg = new BitmapImage();

//theimage = new Image();

//theimage.Stretch = Stretch.Fill;

//theimage.Width = 50;

//theimage.Height = 50;

//theBitImg.BeginInit();

//theBitImg.UriSource = new Uri(@"C:\Users\oskar\Desktop\app dev proj\Application Dev Project\Application Dev Project/b.jpg");

//theBitImg.EndInit();

//theimage.Source = theBitImg;

//gameCanvas.Children.Add(theimage);

bulletsTimer = new DispatcherTimer();

bulletsTimer.Tick += bulletsTimer\_Tick;

bulletsTimer.Interval = new TimeSpan(0, 0, 0, 0, 1);

bulletsTimer.Start();

lettersTimer = new DispatcherTimer();

lettersTimer.Tick += lettersTimer\_Tick;

lettersTimer.Interval = new TimeSpan(0, 0, 0, 0, 100);

// lettersTimer.Start();

gameCanvas.Background = Brushes.Gray;

//initiates the location of a player

thePlayer = new Player(250, 100, gameCanvas);

theBullets = new List<bullets>();

theLetters = new List<letters>();

theNumbers = new List<Number>();

typeOfGame(getTypeOfGame());

}

private void setUserNick(string nick)

{

userNickName = nick;

//throw new NotImplementedException();

}

private string getUserNick()

{

return userNickName;

}

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

{

//checks if the animation can start

if (countToStartSpelling>=1)

{

for (int i = 0; i < theLetters.Count; i++)

{

theLetters[i].move(Directions.random);

if (initiatingLetters == false)

{

//checks if a letter is outside of the canvas

LetterBoundaries(theLetters[i]);

}

}

}

if (countToStartSpelling >= 1 && bulletCount >= 1/\*getLetter.Length\*/)

{

for (int i = 0; i < theBullets.Count; i++)

{

// theLetters[i].move(Directions.random);

for (int j = 0; j < theLetters.Count; j++)

{

if(bulletCount >= 1)

{

//checks for collision between the bullets and the letters

boundaries(theBullets[i], theLetters[j]);

}

}

}

}

//resets the value to zero for every new word

if (countWordsInterv >= 30)

{

countToStartSpelling = 0;

}

}

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

{

theTime++;

letterInterval++;

gameWatch.Text = theTime.ToString();

timeRemainingLbl.Text = timeRemaining.ToString();

nicklbl.Text = getUserNick();

pointstxtB.Text = Convert.ToString(GetScore());

//removes the lletrs from the canvas if the time limit has passed

if (countWordsInterv>=30&&countWordsInterv<=34)

{

for (int i=0;i<theLetters.Count;i++)

{

RemoveLetters(theLetters[i]);

}

}

//Puts a new word to spelled into the game

if (countWordsInterv == 0 || countWordsInterv >= 35/\*&&Wordcompleted==true\*/)

{

//resets the textblocs to 0 for the new coming word

if (timeRemaining < 30)

{

timeRemaining = 30;

letter1.Text = "x";

letter2.Text = "x";

letter3.Text = "x";

letter4.Text = "x";

letter5.Text = "x";

letter6.Text = "x";

letter7.Text = "x";

letter8.Text = "x";

letter9.Text = "x";

letter10.Text = "x";

letter11.Text = "x";

letter12.Text = "x";

letter13.Text = "x";

//timeRemaining = 30;

}

getLetter = listofWords[wordindex];

wordindex++;

countWordsInterv = 0;

}

elementsLbl.Text = getLetter;

countWordsInterv++;

timeRemaining--;

//if (timeRemaining < 0 || reset == true)

//{

//}

if (letterInterval == 1)

{

//restes itself to 0 to allow the words of a new word the be added

if (tempint >= countWordsInterv)

{

tempint = 0;

}

if (tempint <= getLetter.Length - 1)

{

//ads a letter to the canvas until tempint is bigger that the word

AddLetter(getLetter[tempint]);

initiatingLetters = true;

}

else

{

initiatingLetters = false;

}

tempint++;

letterInterval = 0;

}

//incerases to allow the motion of the letters and bullets

countToStartSpelling++;

if (timeRemaining <= 0)

{

letterOrder = 0;

}

checkIfSpellingCorrect();

}

//adds new letter to cnavas

private void AddLetter(char character)

{

letters anyletter = new letters(character, gameCanvas);

// gameCanvas.Children.Add(anyletter);

theLetters.Add(anyletter);

//starts animation timer of to animate letters

lettersTimer.Start();

}

//moves the player

private void theGame\_KeyDown(object sender, KeyEventArgs e)

{

if (e.Key == Key.Up)

{

thePlayer.move(Directions.up);

}

if (e.Key == Key.Down)

{

thePlayer.move(Directions.down);

}

if (e.Key == Key.Left)

{

thePlayer.move(Directions.left);

}

if (e.Key == Key.Right)

{

thePlayer.move(Directions.right);

}

}

internal void setTypeOfGame(string v)

{

theTypeOfGame = v;

}

internal string getTypeOfGame()

{

return theTypeOfGame;

}

private void returnMainMenuBtn\_Click(object sender, RoutedEventArgs e)

{

MainWindow newWindown = new MainWindow();

newWindown.Show();

this.Close();

}

private void endGameBtn\_Click\_1(object sender, RoutedEventArgs e)

{

GameOver = true;

SaveUserDetails(GetScore(),getUserNick());

this.Close();

}

//creates new bullets every time leftButton of the mouse is clicked

private void gameCanvas\_MouseLeftButtonDown(object sender, MouseButtonEventArgs e)

{

bullets bullet = new bullets(gameCanvas, thePlayer.temp3, thePlayer.temp4);

bullet.setAngle(thePlayer.temp1, thePlayer.temp2, thePlayer.temp3, thePlayer.temp4);

fireBullet(bullet);

bulletCount += 1;

//increase everytime bullets if fired

startshooting++;

}

private void fireBullet(bullets b)

{

theBullets.Add(b);

}

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

{

//checks if there is any bullets to be moved

if ( startshooting >= 1)

{

for (int i = 0; i < theBullets.Count; i++)

{

theBullets[i].move(Directions.mouseTrajectory);

bulletBoundaries(theBullets[i]);

}

}

//checks if ther is any bullet in the canvas

//if there is any it checks if it collides with any letter

if (bulletCount >= 1 && countToStartSpelling >= 1/\* getLetter.Length\*/)

{

for (int i = 0; i < theBullets.Count; i++)

{

for (int j = 0; j < theLetters.Count - 1; j++)

{

if (bulletCount>=1)

{

//collision detection

boundaries(theBullets[i], theLetters[j]);

}

}

}

}

//checks if there is a new letter to be spelled if the one before as completely disappeared

if (countWordsInterv >= 30)

{

countToStartSpelling = 0;

}

}

//position of the mouse inside the canvas

private void gameCanvas\_MouseMove\_1(object sender, MouseEventArgs e)

{

mousePosition = e.GetPosition(gameCanvas);

thePlayer.rotate(mousePosition, true);

}

private void gameCanvas\_MouseLeave(object sender, MouseEventArgs e)

{

thePlayer.newPoint = e.GetPosition(gameCanvas);

}

private void typeOfGame(string game)

{

if (game == "Spelling")

{

theSpellingGame = new Thread(playSpelling);

theSpellingGame.Start();

//playSpelling();

//gameCanvas.Children.Add(/\*aLetter\*/);

}

if (game == "Squares")

{

//playSquares();

}

}

private void boundaries(bullets b, letters l)

{

//if (b.shield.X.Equals(l.shield.X)&& b.shield.Y.Equals(l.shield.Y))

//{

// for (int i=0;i<=getLetter.Length-1;i++)

// {

// if (l.letter == getLetter[i] && indexLetter == i)

// {

// }

// }

// indexLetter++;

// bulletCount = 0;

// // CountScore();

// theLetters.Remove(l);

// gameCanvas.Children.Remove(l.therectPath);

// gameCanvas.Children.Remove(l);

// theBullets.Remove(b);

// gameCanvas.Children.Remove(b.theBullet);

// gameCanvas.Children.Remove(b.therectPath);

//}

if (b.shield.IntersectsWith(l.shield))

{

letterOrder++;

DisplayLetters(letterOrder,l);

indexLetter++;

bulletCount = 0;

theLetters.Remove(l);

gameCanvas.Children.Remove(l.therectPath);

gameCanvas.Children.Remove(l);

theBullets.Remove(b);

gameCanvas.Children.Remove(b.theBullet);

gameCanvas.Children.Remove(b.therectPath);

}

}

private void DisplayLetters(int letterOrder, letters l)

{

switch (letterOrder)

{

case 1:

letter1.Text =Convert.ToString( l.letter);

break;

case 2:

letter2.Text = Convert.ToString(l.letter);

break;

case 3:

letter3.Text = Convert.ToString(l.letter);

break;

case 4:

letter4.Text = Convert.ToString(l.letter);

break;

case 5:

letter5.Text = Convert.ToString(l.letter);

break;

case 6:

letter6.Text = Convert.ToString(l.letter);

break;

case 7:

letter7.Text = Convert.ToString(l.letter);

break;

case 8:

letter8.Text = Convert.ToString(l.letter);

break;

case 9:

letter9.Text = Convert.ToString(l.letter);

break;

case 10:

letter10.Text = Convert.ToString(l.letter);

break;

case 11:

letter11.Text = Convert.ToString(l.letter);

break;

case 12:

letter12.Text = Convert.ToString(l.letter);

break;

case 13:

letter13.Text = Convert.ToString(l.letter);

break;

}

// throw new NotImplementedException();

}

private void checkIfSpellingCorrect()

{

int completeTheWord = 13-getLetter.Length;

switch (completeTheWord)

{

case 0:

theWord = getLetter;

break;

case 1:

theWord = getLetter + "x";

break;

case 2:

theWord = getLetter + "xx";

break;

case 3:

theWord = getLetter + "xxx";

break;

case 4:

theWord = getLetter + "xxxx";

break;

case 5:

theWord = getLetter + "xxxxx";

break;

case 6:

theWord = getLetter + "xxxxxx";

break;

case 7:

theWord = getLetter + "xxxxxxx";

break;

}

spelledWord = letter1.Text + letter2.Text + letter3.Text + letter4.Text + letter5.Text + letter6.Text + letter7.Text + letter8.Text + letter9.Text + letter10.Text + letter11.Text + letter12.Text + letter13.Text;

if (spelledWord==theWord)

{

reset = true;

if (reset == true)

{

CountScore();

letter1.Text = "x";

letter2.Text = "x";

letter3.Text = "x";

letter4.Text = "x";

letter5.Text = "x";

letter6.Text = "x";

letter7.Text = "x";

letter8.Text = "x";

letter9.Text = "x";

letter10.Text = "x";

letter11.Text = "x";

letter12.Text = "x";

letter13.Text = "x";

}

}

}

private void bulletBoundaries(bullets b)

{

if (b.shield.X >= gameCanvas.ActualWidth-10 || b.shield.X <= 1)

{

theBullets.Remove(b);

gameCanvas.Children.Remove(b.theBullet);

gameCanvas.Children.Remove(b.therectPath);

bulletCount = 0;

}

else if (b.shield.Y >= gameCanvas.ActualHeight-10 || b.shield.Y <= 1)

{

theBullets.Remove(b);

gameCanvas.Children.Remove(b.theBullet);

gameCanvas.Children.Remove(b.therectPath);

bulletCount = 0;

}

}

private void LetterBoundaries(letters l)

{

if (l.shield.X >= gameCanvas.ActualWidth-50 || l.shield.X <= 1)

{

gameCanvas.Children.Remove(l.therectPath);

gameCanvas.Children.Remove(l);

theLetters.Remove(l);

}

else if (l.shield.Y >= gameCanvas.ActualHeight-50 || l.shield.Y <= 1)

{

gameCanvas.Children.Remove(l.therectPath);

gameCanvas.Children.Remove(l);

theLetters.Remove(l);

}

}

private void CountScore()

{

realSCore++;

}

private int GetScore()

{

return realSCore;

}

void playSpelling()

{

listofWords = new List<string>();

GetTheletters();

gameTimer.Start();

}

//get the 50 words from the database and adds it to a list of letters

private void GetTheletters()

{

int count = 0;

listofWords.Clear();

OleDbConnection connectionToWords = WordsDBConnection();

string wordQuery = "";

wordQuery = "SELECT \* FROM student";

OleDbCommand myCommand = new OleDbCommand(wordQuery, connectionToWords);

try

{

connectionToWords.Open();

OleDbDataReader wordReader = myCommand.ExecuteReader();

while (wordReader.Read())

{

listofWords.Add(wordReader["fullName"].ToString());

count++;

}

}

catch (Exception ex)

{

MessageBox.Show("Exception in DBHandler" + ex);

}

finally

{

connectionToWords.Close();

}

}

//private void playSquares()

//{

//}

private static OleDbConnection WordsDBConnection()

{

String Wordsconn = @"Provider = Microsoft.JET.OLEDB.4.0;Data Source = C:\Users\oskar\Documents\App Dev Project\Application Dev Project\Application Dev Project\cwDBExample.mdb";

return new OleDbConnection(Wordsconn);

}

private void CheckIfGameOver()

{

if (theTime == 1500)

{

GameOver = true;

}

}

//updated the players score when he decides to leave the game

private void SaveUserDetails(int userscore, string theNickName)

{

OleDbConnection playerData = WordsDBConnection();

OleDbCommand updateComand = new OleDbCommand();

updateComand.CommandType = CommandType.Text;

updateComand.CommandText = "UPDATE playerDetails SET userName = @userName, score = @score WHERE userName = @userName";

updateComand.Parameters.AddWithValue("@userName", theNickName);

updateComand.Parameters.AddWithValue("@score", userscore);

updateComand.Connection = playerData;

try

{

playerData.Open();

updateComand.ExecuteNonQuery();

}

catch (Exception ex)

{

MessageBox.Show("Exception in DBHandler" + ex);

}

finally

{

playerData.Close();

}

}

private void RemoveLetters(letters l)

{

gameCanvas.Children.Remove(l.therectPath);

gameCanvas.Children.Remove(l);

theLetters.Remove(l);

}

}

}

namespace Application\_Dev\_Project

{

public partial class UserName : Window, IGameEngine

{

private string uName;

public bool safeThreadClosing;

private bool check = false;

private string typeOfGame;

public UserName()

{

InitializeComponent();

WindowStartupLocation = System.Windows.WindowStartupLocation.CenterScreen;

safeThreadClosing = false;

}

//gets the user nickname and type of game that wants to be played

private void getUNbtn\_Click(object sender, RoutedEventArgs e)

{

uName = userameNick.Text;

checkUName(uName);

}

private void checkUName(string name)

{

if (name == "" || name == " ")

{

MessageBox.Show("Incorrect nickname. Nickname can not be empty");

}

else if (typeOfGame == "Squares")

{

MessageBox.Show("Unavailable Option. Try the Spelling(available) option");

}

else

{

check = true;

SaveUserDetails(name);

windowOpener newPlayingWindow = new windowOpener(typeOfWindow.gameScreen, typeOfGame,name);

safeThreadClosing = true;

this.Close();

}

}

public void move(Directions move)

{

}

public void rotate(Point location, bool checkmouseInScreen)

{

// throw new NotImplementedException();

}

private void SpellingRbtn\_Checked(object sender, RoutedEventArgs e)

{

typeOfGame = "Spelling";

}

private void SquaresRbtn\_Checked(object sender, RoutedEventArgs e)

{

typeOfGame = "Squares";

}

private void SaveUserDetails(string usernick)

{

OleDbConnection connectionToWords = WordsDBConnection();

string wordQuery = "";

wordQuery = "INSERT INTO playerDetails( userName) VALUES ( '" +usernick+ "' )";

OleDbCommand myCommand = new OleDbCommand(wordQuery, connectionToWords);

try

{

connectionToWords.Open();

myCommand.ExecuteNonQuery();

}

catch (Exception ex)

{

MessageBox.Show("Exception in DBHandler" + ex);

}

finally

{

connectionToWords.Close();

}

}

private static OleDbConnection WordsDBConnection()

{

String Wordsconn = @"Provider = Microsoft.JET.OLEDB.4.0;Data Source = C:\Users\oskar\Documents\App Dev Project\Application Dev Project\Application Dev Project\cwDBExample.mdb";

return new OleDbConnection(Wordsconn);

}

public bool Continue()

{

bool theanswer=check;

return theanswer;

}

}

}

namespace Application\_Dev\_Project

{

//interface to make the bodies move

interface IGameEngine

{

void move(Directions move);

void rotate(Point location, bool checkmouseInScreen);

}

}

namespace Application\_Dev\_Project

{

public enum Directions

{

//trajectory direction

up, down, right, left, mouseTrajectory, random

}

}

namespace Application\_Dev\_Project

{

class bullets : movingObject

{

private double X { get; set; }//tracks the x cordinate of the bullet's location

private double Y { get; set; }//tracks the y cordinate of the bullet's location

public Ellipse theBullet = new Ellipse();//stores the last location of the bullets

public Path therectPath = new Path();//the rectangle around the bullets

public Canvas bulletCanvas = new Canvas();

private bool isFired = false;//checks if bullet can move

private double velocityx = 0;//speed in the x axes

private double velocityy = 0;//speed in the y axes

private double angle = 0;//angle of ratation of the fireArm

public Rect shield = new Rect();//the rectangle around the bullet

public bullets(Canvas c, double x, double y)

{

X = x;

Y = y;

bulletCanvas = c;

theBullet = bullet();//adds bullets to the canvas

}

//Creating the body of the bullet

public Ellipse bullet()

{

TranslateTransform bulletTranslate = new TranslateTransform();

TransformGroup bulletAsALineTransGroup = new TransformGroup();

GeometryGroup gG = new GeometryGroup();

Ellipse bulletAsEllipse = new Ellipse();

Path rectPath = new Path();

rectPath.Stroke = Brushes.Black;

rectPath.StrokeThickness = 1;

shield.Width = bulletAsEllipse.Width = 10;

shield.Height = bulletAsEllipse.Height = 10;

SolidColorBrush sb = new SolidColorBrush(Colors.Red);

bulletAsEllipse.Fill = sb;

//specify its initial location

bulletAsEllipse.Margin = new Thickness(X - 5, Y - 5, 0, 0);

//specify its initial location

shield.X = X - 5;

shield.Y = Y - 5;

if (isFired == true)

{

//make the bullet move

bulletTranslate.X = bulletTranslate.X + velocityx;

bulletTranslate.Y = bulletTranslate.Y - velocityy;

shield.X = shield.X + velocityx;

shield.Y = shield.Y - velocityy;

}

RectangleGeometry rG = new RectangleGeometry();

rG.Rect = shield;

gG.Children.Add(rG);

rectPath.Data = gG;

therectPath = rectPath;

bulletAsALineTransGroup.Children.Add(bulletTranslate);

bulletAsEllipse.RenderTransform = bulletAsALineTransGroup;

//Adds bullets to the canvas

bulletCanvas.Children.Add(bulletAsEllipse);

bulletCanvas.Children.Add(rectPath);

return bulletAsEllipse;

}

public override void move(Directions move)

{

isFired = true;

//Deletes the previous bullet

bulletCanvas.Children.Remove(theBullet);

bulletCanvas.Children.Remove(therectPath);

if (move == Directions.mouseTrajectory)

{

velocityx = velocityx + (8 \* Math.Cos(getAngle()));

velocityy = velocityy + (8 \* Math.Sin(getAngle()));

}

// bullet redrawn

theBullet = bullet();

}

//the agle between the cursor and the body of the location of the character

public void setAngle(double x1, double y1, double x2, double y2)

{

double numb = x2 - x1;

double numb2 = y1 - y2;

double newang = Math.Atan2(numb2, numb);

angle = newang;

}

private double getAngle()

{

return angle;

}

public override void rotate(Point location, bool checkmouseInScreen)

{

}

}

}

namespace Application\_Dev\_Project

{

class letters :System.Windows.Controls.Image, IGameEngine

{

Canvas letterCanvas = new Canvas();

public System.Windows.Shapes.Path therectPath = new System.Windows.Shapes.Path();

// double angle = 0;

double velocityx = 0; //x component of speed

double velocityy = 0;//x component of speed

double velx = 0;// x component of speed

double vely = 0;// x component of speed

public char letter;//gets the letter

public double X { get; set; }//tracks the x cordinate of the letter's location

public double Y { get; set; }//tracks the y cordinate of the letter's location

BitmapImage imagePath;

public Rect shield = new Rect();

public letters(char charac, Canvas c)

{

letterCanvas = c;

letter = charac;

string path = @"C:\Users\oskar\Documents\App Dev Project\Application Dev Project\Application Dev Project\" + charac+".jpg";

imagePath = new BitmapImage();

this.Stretch = Stretch.Fill;

shield.Width = this.Width = 30;

shield.Height=this.Height = 30;

appear();

imagePath.BeginInit();

imagePath.UriSource = new Uri(path,UriKind.RelativeOrAbsolute);

imagePath.EndInit();

this.Source = imagePath;

letterCanvas.Children.Add(this);

}

public void move(Directions move)

{

//removes letter from previous location

letterCanvas.Children.Remove(therectPath);

letterCanvas.Children.Remove(this);

TranslateTransform letterTranslate = new TranslateTransform();

TransformGroup letterTransGroup = new TransformGroup();

GeometryGroup gG = new GeometryGroup();

System.Windows.Shapes.Path rectPath = new System.Windows.Shapes.Path();

rectPath.Stroke = Brushes.Black;

rectPath.StrokeThickness = 1;

if (move == Directions.random)

{

//letters move

velocityx = velocityx + velx;

velocityy = velocityy + vely;

letterTranslate.X = letterTranslate.X + velocityx;

letterTranslate.Y = letterTranslate.Y + velocityy;

X = X + velocityx;

Y = Y + velocityy;

shield.X = shield.X+velx;

shield.Y = shield.Y+ +vely;

}

RectangleGeometry rG = new RectangleGeometry();

rG.Rect = shield;

gG.Children.Add(rG);

rectPath.Data = gG;

therectPath = rectPath;

letterTransGroup.Children.Add(letterTranslate);

this.RenderTransform = letterTransGroup;

//adds letter into new location

letterCanvas.Children.Add(rectPath);

letterCanvas.Children.Add(this);

}

public void rotate(System.Windows.Point location, bool checkmouseInScreen)

{

}

//each new letter shows from a different random location

private void appear()

{

Random side= new Random();

Random placex = new Random();

Random placey = new Random();

double x = placex.Next(50, Convert.ToInt16(letterCanvas.ActualWidth-50));

double y = placey.Next(50, Convert.ToInt16(letterCanvas.ActualHeight - 50));

int sidechoice = 0;

Directions theside = new Directions();

sidechoice = side.Next(0,4);

if (sidechoice == 0)

{

theside = Directions.up;

this.Margin= new Thickness(x,0,0,0);

velx = 0;

vely = 5;

X = x;

Y = 0;

shield.X = x;

shield.Y = 0;

}

else if (sidechoice==1)

{

theside = Directions.down;

this.Margin = new Thickness(x, letterCanvas.ActualHeight, 0, 0);

velx = 0;vely = -5;

X = x;

Y = letterCanvas.ActualHeight;

shield.X = x;

shield.Y = letterCanvas.ActualHeight;

}

else if (sidechoice == 2)

{

theside = Directions.right;

this.Margin = new Thickness(0, y, 0, 0);

velx = 5;

vely = 0;

X = 0;

Y = y;

shield.X = 0;

shield.Y = y;

}

else if (sidechoice == 3)

{

theside = Directions.left;

this.Margin = new Thickness(letterCanvas.ActualWidth, y, 0, 0);

velx = -5;

vely = 0;

X = letterCanvas.ActualWidth;

Y = y;

shield.X = letterCanvas.ActualWidth;

shield.Y = y;

}

}

}

}

namespace Application\_Dev\_Project {

/// <summary>

/// playGameScreen

/// </summary>

public partial class playGameScreen : System.Windows.Window, System.Windows.Markup.IComponentConnector {

#line 1 "..\..\playGameScreen.xaml"

[System.Diagnostics.CodeAnalysis.SuppressMessageAttribute("Microsoft.Performance", "CA1823:AvoidUnusedPrivateFields")]

internal Application\_Dev\_Project.playGameScreen theGame;

#line default

#line hidden

#line 11 "..\..\playGameScreen.xaml"

[System.Diagnostics.CodeAnalysis.SuppressMessageAttribute("Microsoft.Performance", "CA1823:AvoidUnusedPrivateFields")]

internal System.Windows.Controls.Canvas gameCanvas;

#line default

#line hidden

#line 12 "..\..\playGameScreen.xaml"

[System.Diagnostics.CodeAnalysis.SuppressMessageAttribute("Microsoft.Performance", "CA1823:AvoidUnusedPrivateFields")]

internal System.Windows.Controls.Button returnMainMenuBtn;

#line default

#line hidden

#line 13 "..\..\playGameScreen.xaml"

[System.Diagnostics.CodeAnalysis.SuppressMessageAttribute("Microsoft.Performance", "CA1823:AvoidUnusedPrivateFields")]

internal System.Windows.Controls.Button endGameBtn;

#line default

#line hidden

#line 14 "..\..\playGameScreen.xaml"

[System.Diagnostics.CodeAnalysis.SuppressMessageAttribute("Microsoft.Performance", "CA1823:AvoidUnusedPrivateFields")]

internal System.Windows.Controls.Button pauseResumeBtn;

#line default

#line hidden

#line 15 "..\..\playGameScreen.xaml"

[System.Diagnostics.CodeAnalysis.SuppressMessageAttribute("Microsoft.Performance", "CA1823:AvoidUnusedPrivateFields")]

internal System.Windows.Controls.TextBlock gameWatch;

#line default

#line hidden

#line 16 "..\..\playGameScreen.xaml"

[System.Diagnostics.CodeAnalysis.SuppressMessageAttribute("Microsoft.Performance", "CA1823:AvoidUnusedPrivateFields")]

internal System.Windows.Controls.TextBlock elementsLbl;

#line default

#line hidden

#line 18 "..\..\playGameScreen.xaml"

[System.Diagnostics.CodeAnalysis.SuppressMessageAttribute("Microsoft.Performance", "CA1823:AvoidUnusedPrivateFields")]

internal System.Windows.Controls.TreeView treeView1;

#line default

#line hidden

#line 20 "..\..\playGameScreen.xaml"

[System.Diagnostics.CodeAnalysis.SuppressMessageAttribute("Microsoft.Performance", "CA1823:AvoidUnusedPrivateFields")]

internal System.Windows.Controls.TextBlock timeRemainingLbl;

#line default

#line hidden

#line 21 "..\..\playGameScreen.xaml"

[System.Diagnostics.CodeAnalysis.SuppressMessageAttribute("Microsoft.Performance", "CA1823:AvoidUnusedPrivateFields")]

internal System.Windows.Controls.TextBlock letter1;

#line default

#line hidden

#line 22 "..\..\playGameScreen.xaml"

[System.Diagnostics.CodeAnalysis.SuppressMessageAttribute("Microsoft.Performance", "CA1823:AvoidUnusedPrivateFields")]

internal System.Windows.Controls.TextBlock letter2;

#line default

#line hidden

#line 23 "..\..\playGameScreen.xaml"

[System.Diagnostics.CodeAnalysis.SuppressMessageAttribute("Microsoft.Performance", "CA1823:AvoidUnusedPrivateFields")]

internal System.Windows.Controls.TextBlock letter3;

#line default

#line hidden

#line 24 "..\..\playGameScreen.xaml"

[System.Diagnostics.CodeAnalysis.SuppressMessageAttribute("Microsoft.Performance", "CA1823:AvoidUnusedPrivateFields")]

internal System.Windows.Controls.TextBlock letter4;

#line default

#line hidden

#line 25 "..\..\playGameScreen.xaml"

[System.Diagnostics.CodeAnalysis.SuppressMessageAttribute("Microsoft.Performance", "CA1823:AvoidUnusedPrivateFields")]

internal System.Windows.Controls.TextBlock letter5;

#line default

#line hidden

#line 26 "..\..\playGameScreen.xaml"

[System.Diagnostics.CodeAnalysis.SuppressMessageAttribute("Microsoft.Performance", "CA1823:AvoidUnusedPrivateFields")]

internal System.Windows.Controls.TextBlock letter7;

#line default

#line hidden

#line 27 "..\..\playGameScreen.xaml"

[System.Diagnostics.CodeAnalysis.SuppressMessageAttribute("Microsoft.Performance", "CA1823:AvoidUnusedPrivateFields")]

internal System.Windows.Controls.TextBlock letter8;

#line default

#line hidden

#line 28 "..\..\playGameScreen.xaml"

[System.Diagnostics.CodeAnalysis.SuppressMessageAttribute("Microsoft.Performance", "CA1823:AvoidUnusedPrivateFields")]

internal System.Windows.Controls.TextBlock letter9;

#line default

#line hidden

#line 29 "..\..\playGameScreen.xaml"

[System.Diagnostics.CodeAnalysis.SuppressMessageAttribute("Microsoft.Performance", "CA1823:AvoidUnusedPrivateFields")]

internal System.Windows.Controls.TextBlock letter10;

#line default

#line hidden

#line 30 "..\..\playGameScreen.xaml"

[System.Diagnostics.CodeAnalysis.SuppressMessageAttribute("Microsoft.Performance", "CA1823:AvoidUnusedPrivateFields")]

internal System.Windows.Controls.TextBlock letter11;

#line default

#line hidden

#line 31 "..\..\playGameScreen.xaml"

[System.Diagnostics.CodeAnalysis.SuppressMessageAttribute("Microsoft.Performance", "CA1823:AvoidUnusedPrivateFields")]

internal System.Windows.Controls.TextBlock letter12;

#line default

#line hidden

#line 32 "..\..\playGameScreen.xaml"

[System.Diagnostics.CodeAnalysis.SuppressMessageAttribute("Microsoft.Performance", "CA1823:AvoidUnusedPrivateFields")]

internal System.Windows.Controls.TextBlock letter6;

#line default

#line hidden

#line 33 "..\..\playGameScreen.xaml"

[System.Diagnostics.CodeAnalysis.SuppressMessageAttribute("Microsoft.Performance", "CA1823:AvoidUnusedPrivateFields")]

internal System.Windows.Controls.TextBlock letter13;

#line default

#line hidden

#line 34 "..\..\playGameScreen.xaml"

[System.Diagnostics.CodeAnalysis.SuppressMessageAttribute("Microsoft.Performance", "CA1823:AvoidUnusedPrivateFields")]

internal System.Windows.Controls.TextBlock pointstxtB;

#line default

#line hidden

#line 35 "..\..\playGameScreen.xaml"

[System.Diagnostics.CodeAnalysis.SuppressMessageAttribute("Microsoft.Performance", "CA1823:AvoidUnusedPrivateFields")]

internal System.Windows.Controls.TextBlock nicklbl;

#line default

#line hidden

private bool \_contentLoaded;

/// <summary>

/// InitializeComponent

/// </summary>

[System.Diagnostics.DebuggerNonUserCodeAttribute()]

[System.CodeDom.Compiler.GeneratedCodeAttribute("PresentationBuildTasks", "4.0.0.0")]

public void InitializeComponent() {

if (\_contentLoaded) {

return;

}

\_contentLoaded = true;

System.Uri resourceLocater = new System.Uri("/Application Dev Project;component/playgamescreen.xaml", System.UriKind.Relative);

#line 1 "..\..\playGameScreen.xaml"

System.Windows.Application.LoadComponent(this, resourceLocater);

#line default

#line hidden

}

[System.Diagnostics.DebuggerNonUserCodeAttribute()]

[System.CodeDom.Compiler.GeneratedCodeAttribute("PresentationBuildTasks", "4.0.0.0")]

[System.ComponentModel.EditorBrowsableAttribute(System.ComponentModel.EditorBrowsableState.Never)]

[System.Diagnostics.CodeAnalysis.SuppressMessageAttribute("Microsoft.Design", "CA1033:InterfaceMethodsShouldBeCallableByChildTypes")]

[System.Diagnostics.CodeAnalysis.SuppressMessageAttribute("Microsoft.Maintainability", "CA1502:AvoidExcessiveComplexity")]

[System.Diagnostics.CodeAnalysis.SuppressMessageAttribute("Microsoft.Performance", "CA1800:DoNotCastUnnecessarily")]

void System.Windows.Markup.IComponentConnector.Connect(int connectionId, object target) {

switch (connectionId)

{

case 1:

this.theGame = ((Application\_Dev\_Project.playGameScreen)(target));

#line 8 "..\..\playGameScreen.xaml"

this.theGame.KeyDown += new System.Windows.Input.KeyEventHandler(this.theGame\_KeyDown);

#line default

#line hidden

return;

case 2:

this.gameCanvas = ((System.Windows.Controls.Canvas)(target));

#line 11 "..\..\playGameScreen.xaml"

this.gameCanvas.MouseLeftButtonDown += new System.Windows.Input.MouseButtonEventHandler(this.gameCanvas\_MouseLeftButtonDown);

#line default

#line hidden

#line 11 "..\..\playGameScreen.xaml"

this.gameCanvas.MouseMove += new System.Windows.Input.MouseEventHandler(this.gameCanvas\_MouseMove\_1);

#line default

#line hidden

#line 11 "..\..\playGameScreen.xaml"

this.gameCanvas.MouseLeave += new System.Windows.Input.MouseEventHandler(this.gameCanvas\_MouseLeave);

#line default

#line hidden

return;

case 3:

this.returnMainMenuBtn = ((System.Windows.Controls.Button)(target));

#line 12 "..\..\playGameScreen.xaml"

this.returnMainMenuBtn.Click += new System.Windows.RoutedEventHandler(this.returnMainMenuBtn\_Click);

#line default

#line hidden

return;

case 4:

this.endGameBtn = ((System.Windows.Controls.Button)(target));

#line 13 "..\..\playGameScreen.xaml"

this.endGameBtn.Click += new System.Windows.RoutedEventHandler(this.endGameBtn\_Click\_1);

#line default

#line hidden

return;

case 5:

this.pauseResumeBtn = ((System.Windows.Controls.Button)(target));

return;

case 6:

this.gameWatch = ((System.Windows.Controls.TextBlock)(target));

return;

case 7:

this.elementsLbl = ((System.Windows.Controls.TextBlock)(target));

return;

case 8:

this.treeView1 = ((System.Windows.Controls.TreeView)(target));

return;

case 9:

this.timeRemainingLbl = ((System.Windows.Controls.TextBlock)(target));

return;

case 10:

this.letter1 = ((System.Windows.Controls.TextBlock)(target));

return;

case 11:

this.letter2 = ((System.Windows.Controls.TextBlock)(target));

return;

case 12:

this.letter3 = ((System.Windows.Controls.TextBlock)(target));

return;

case 13:

this.letter4 = ((System.Windows.Controls.TextBlock)(target));

return;

case 14:

this.letter5 = ((System.Windows.Controls.TextBlock)(target));

return;

case 15:

this.letter7 = ((System.Windows.Controls.TextBlock)(target));

return;

case 16:

this.letter8 = ((System.Windows.Controls.TextBlock)(target));

return;

case 17:

this.letter9 = ((System.Windows.Controls.TextBlock)(target));

return;

case 18:

this.letter10 = ((System.Windows.Controls.TextBlock)(target));

return;

case 19:

this.letter11 = ((System.Windows.Controls.TextBlock)(target));

return;

case 20:

this.letter12 = ((System.Windows.Controls.TextBlock)(target));

return;

case 21:

this.letter6 = ((System.Windows.Controls.TextBlock)(target));

return;

case 22:

this.letter13 = ((System.Windows.Controls.TextBlock)(target));

return;

case 23:

this.pointstxtB = ((System.Windows.Controls.TextBlock)(target));

return;

case 24:

this.nicklbl = ((System.Windows.Controls.TextBlock)(target));

return;

}

this.\_contentLoaded = true;

}

}

}

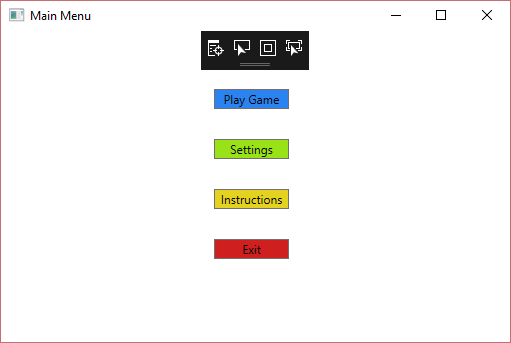
## **2.2. How does it work?**

When the application is executed, a main menu appears in the centre of the screen displaying four options for the user to choose. The user must choose the “Play game” option and when it does another window asking for the player’s details (nickname) will show up. The window also gives the option to select the type of game that is going to be played. Either spelling a word or recognizing square numbers (Only the earlier option works). The program will check if your nickname format is correct and the game will start. The player and the word to be spelled will appear in the screen and seconds later, letters will emerge from different parts of the window. The goal is to shoot all the letters in a desired order (same order as they are in the word). Every time a letter is hit by a bullet, it disappears, and the letter is displayed on the left-hand side of the window in a sequence. If the sequence of letters on the left is equal to the word that has to be spelled points are added. There are 50 words in total and there is 30 seconds to shoot all letters and 5 seconds more to get ready for the next word. The player has the option to return to the main menu or to exit the game.

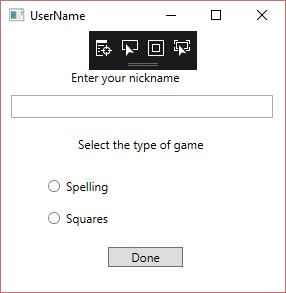
In order to shoot the player must click the left button of the mouse. The character gun points to the location of the mouse (if the cursor is inside the playing screen) which give the notion that the gun is rotating. The player can move and shoot at same time.

## **3.2. Screenshots**

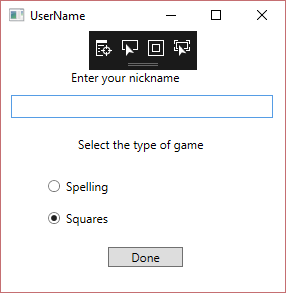
This is the first windows that appears when the application starts running. This is the main menu of the game indicating the options that the player ha to choose from. Choosing “Play Game” will take you to the game.

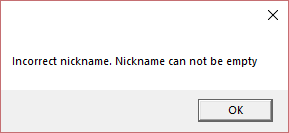


When the “Play Game” option is selected, another window appears asking for the player to enter their user name and giving the user the options to choose between different types of games.

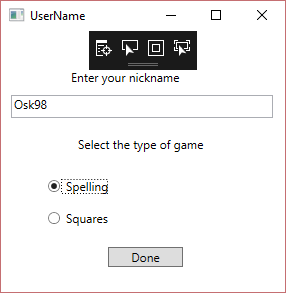


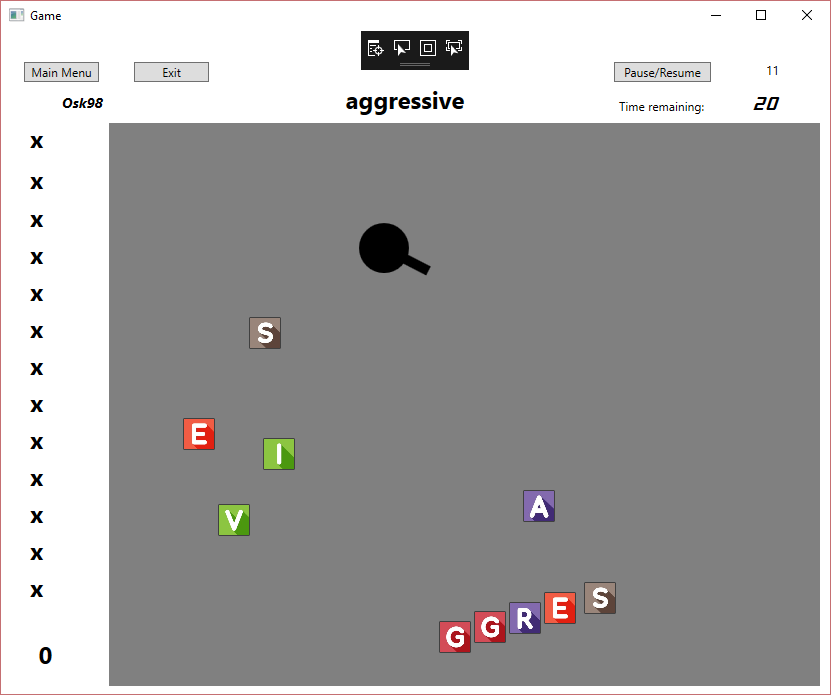
If the nickname inputted does not follow the required format or the type of game selected is not “Spelling” , the same window will reappear for you to enter the right username and select the correct option. When everything is correct the game will start.



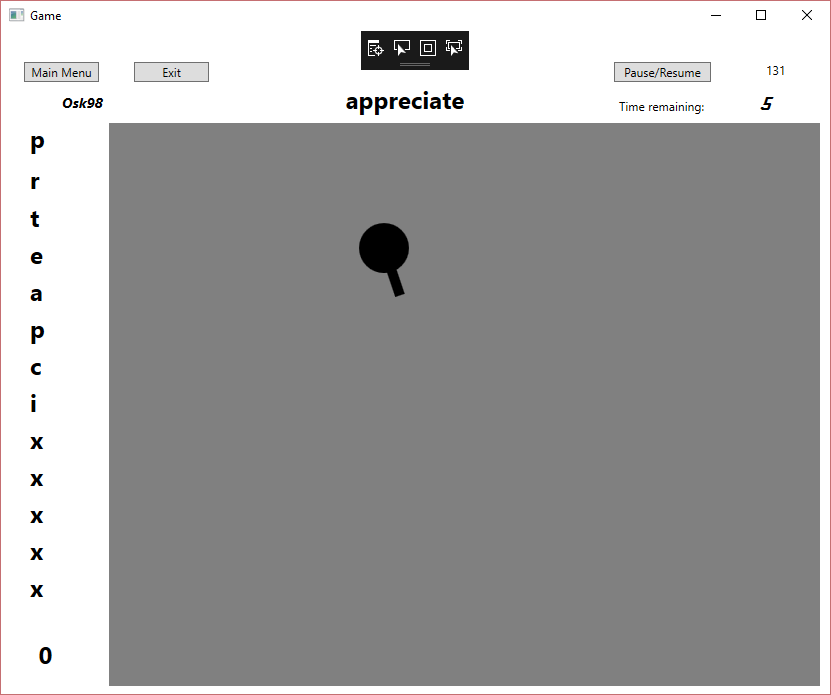


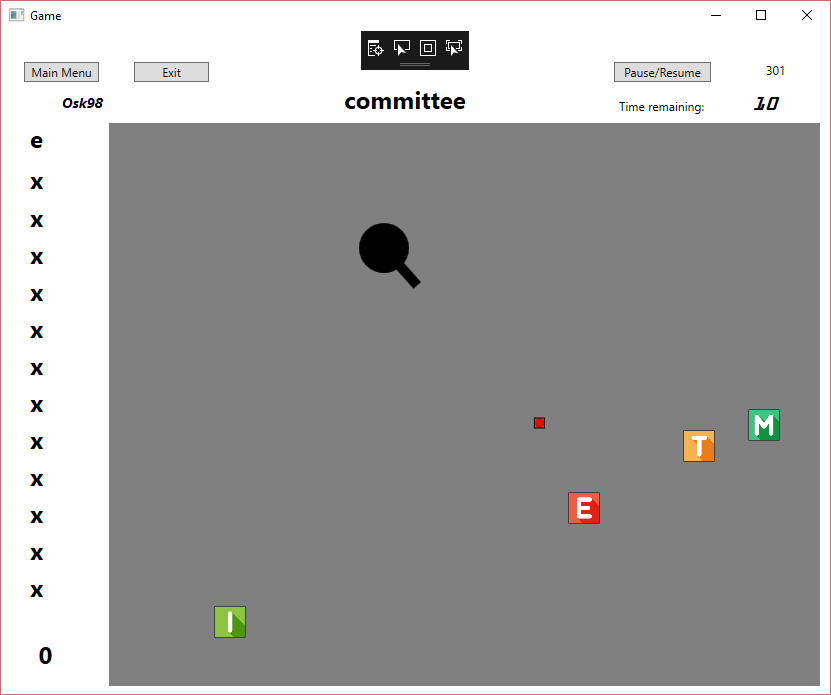
When the game starts the character appears in a position on the screen. As well as the player’s nickname, two timers, three buttons, the words to be spelled, the score(at the bottom) and empty cells on the left that will be filled with the letters that are being hit by the bullets. One timer is to display time that has passed while the game is being played and the other is to show the time remaining for the appearance of the next word. The letters that comprise the words will also appear from random places.

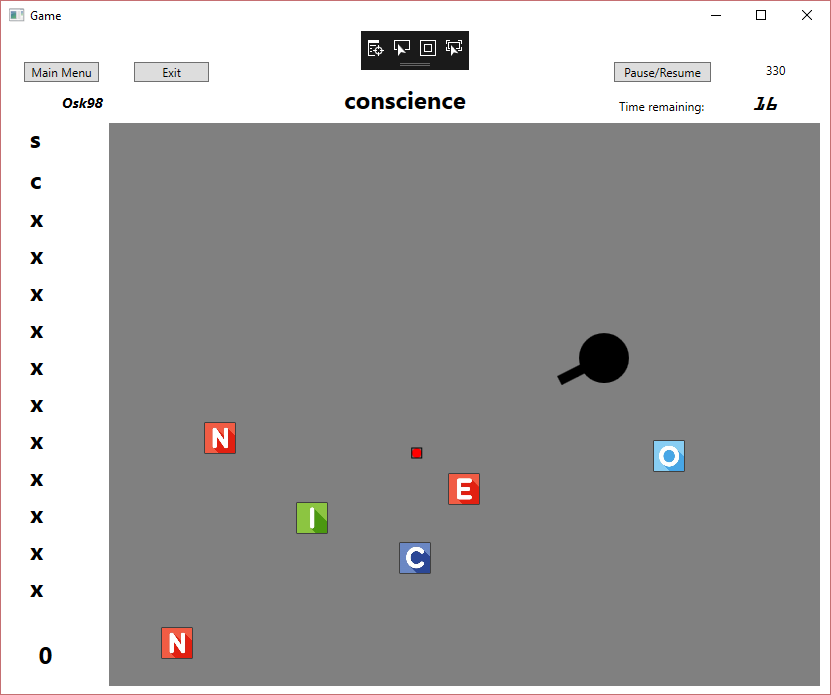




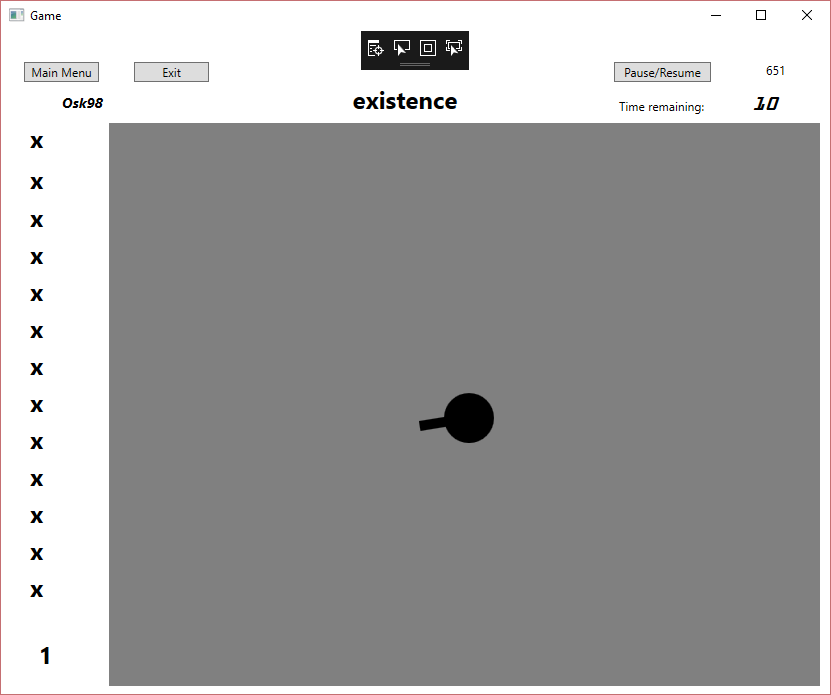
The letters will travel throughout the screen and if any of them is not, they will keep moving until they disappear from the window which means that no points will be awarded, and the player will have to wait for the next word to come up.



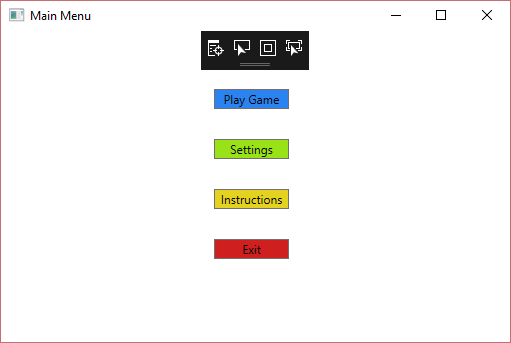




When a word id spelled correctly the score at the bottom will increase and the cells will be reset to their original value again.



The player has to spell correctly as many word as possible. The player can go back to the main menu by clicking on the according button or can leave the game by pressing button “Exit”. When either button is pressed, the score of the player will be saved with their nickname.



## **3.4. Self-Assessment Sheet**

NOTE: You must include comments for each section of the self-assessment sheet Student Name: Oscar Eko Osa

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | % | P | A | S | G | E | O | Comments |
| Object Oriented Design (including UML diagrams and how the design is reflected into code and any other technical documentation included in the report) | 15 |  |  |  | x |  |  | I have done the UML diagrams and the class diagrams although the UML diagram could have been done better. The description of my design could improve by focusing more on the design rather than what inspired the design |
| Use of desired features and |  |  |  |  |  | x | x  x  x | At the begging of every class there are variables (int X,Y) and properties. Methods are in the body most of classes (void move). Decisions are made when the keys are pressed to move the character. Iterations are used when checking the player’s nickname and type of game.  I have player, bullet and letter class and their objects are created I the screen where the game is played. Bullets and letters are stored in collection.  Events are used when the characters gun follows the mouse (playing window) and when keys are pressed (playing window). Validation is used in the “windowOpener” class to check what type of window to open  A thread is used when the game starts to move the letters |
| concepts |  |
| * Well named variables and, |  |
| methods. Decisions, Iteration. Overloading | 10 |
| * Objects and Classes. Collections | 10 |
| * Events controls. Validation | 10 |
| * Threads. Animations. | 10 |

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| * Inheritance. Interfaces * Persistence – storing and reading data from a db * Patterns * Extra features ( such using web services) | 10  10  5  5 |  |  |  |  |  | x  x | IGameEngine id interface used by the player class and the Letter class. The letter class also inherits from the class Image to add the pictures of the alphabet letters  It is UserName class when the nickname and the type of game are entered also in the playingScreen class to update the player details of he decides to close the game.  Not used  Not used |
| Acceptance Testing (includes how well the app is presented and the ability to answer technical questions) | 10 |  |  |  | x |  |  | The design and presentation of the game and windows could have been better |
| Accurate Self- Assessment | 5 |  |  |  |  |  | x | I have been objective |
| Possible AO? |  |  |  |  |  |  |  |  |

# **REFERENCES**

* (Stackoverflow) insert, delete and update in C# using database? [closed]. Available from: <https://stackoverflow.com/questions/15578758/insert-delete-and-update-in-c-sharp-using-database#15578811> [Accessed on the 2 December of 2018]
* (Freepick) Colorful alphabet in square Free Vector. Available from: <https://www.freepik.com/free-vector/colorful-alphabet-in-square_958007.htm#term=alphabet&page=1&position=18> [Accessed on the 21st of November of 2018]
* (Stackoverflow) How do I create and show WPF windows on separate threads? Available from: <https://stackoverflow.com/questions/1111369/how-do-i-create-and-show-wpf-windows-on-separate-threads> [Accessed on the 21st of November of 2018]
* (Dummies) HOW TO FIND VECTOR COMPONENTS. Available from: <https://www.dummies.com/education/science/physics/how-to-find-vector-components/> [Accessed on the 10th of November 2019]