Iterative development of Breakout using Object-Oriented Programming (OOP)

This document is to help you understand the iterative development process and learn games programming techniques in C# that will be of use in your own projects. Note: no-one will be allowed to choose Breakout as their A Level Computer Science Project as I have used it in the support materials.

The analysis of the problem produces the initial outline success criteria and requirements. This document will focus on how you go from an outline set of requirements, through iterative development to a working solution that can then be evaluated.

The stakeholders involved in this project are Lisa and Jay. They are both 17 year old students with an interest in retro arcade games.

Contents

[Outline Requirements for Breakout 2](#_Toc454129872)

[Hardware and Software Configuration 2](#_Toc454129873)

[Iteration 1 – (Static) Game Object 3](#_Toc454129874)

[Discussion with stakeholders 3](#_Toc454129875)

[Specify the proposed solution – Game Objects Detailed Requirements 3](#_Toc454129876)

[Decomposing the problem 3](#_Toc454129877)

[Describe the Solution - Usability features 3](#_Toc454129878)

[Describe the approach to testing – Usability Testing Plan 4](#_Toc454129879)

[Describe the Solution 5](#_Toc454129880)

[Describe the approach to testing 5](#_Toc454129881)

[During Development Testing Plan 5](#_Toc454129882)

[Post Development System Testing for Function and Robustness Plan 6](#_Toc454129883)

[Describe the solution – Pseudo code algorithms and justification 8](#_Toc454129884)

[Developing the solution – Coding 9](#_Toc454129885)

[Testing to inform development 10](#_Toc454129886)

[Stakeholder review and sign-off of iteration 14](#_Toc454129887)

[Iteration 2 – Moveable Game Object 14](#_Toc454129888)

[Discussion with Stakeholders 14](#_Toc454129889)

[Specify the proposed solution – Moveable Game Objects Detailed Requirements 14](#_Toc454129890)

[Decompose the problem 15](#_Toc454129891)

[Describe the Solution - Usability features 15](#_Toc454129892)

[Describe the approach to testing – Usability Testing Plan 15](#_Toc454129893)

[Describe the Solution 16](#_Toc454129894)

[Describe the approach to testing 17](#_Toc454129895)

[During Development Testing Plan 17](#_Toc454129896)

[Post Development System Testing for Function and Robustness Plan 17](#_Toc454129897)

[Describe the solution – Pseudo code algorithms and justification 18](#_Toc454129898)

[Developing the solution – Coding 20](#_Toc454129899)

[Testing to inform development 21](#_Toc454129900)

[Stakeholder review and sign-off of iteration 22](#_Toc454129901)

[Final code to get game working 24](#_Toc454129902)

Outline Requirements for Breakout  
Game screen, objects and movement  
The Breakout game screen will consist of 3 walls – left, right and top.  
There must be a paddle which can be moved left and right.  
There must be a set of bricks with each individual brick being visible or invisible.  
There must be a ball which can be bounced off the paddle, left, right and top wall.  
The ball must pass through invisible bricks and when it hits a visible brick this brick must disappear and the ball bounce (change direction).  
  
Game play  
The human player controls the paddle.   
When the human starts the game the ball is released.  
There will be a finite number of lives in one game.  
A life is lost when the ball misses the paddle and disappears beneath.  
When all bricks have been hit by the ball the human has successfully completed the game.

## Hardware and Software Configuration

The recommended hardware requirements are based on the .NET Framework System Requirements as this will be needed in order for my application to run.  
Processor 1GHz  
RAM 512 MB  
Disk Space 4.6GB  
Input devices mouse and keyboard  
Monitor with a resolution that can accommodate the game screen of 640 x 482 pixels.

The recommended software requirements are:  
Operating system Windows 7 SP1   
.NET Framework 4.5

# Iteration 1 – (Static) Game Object

The game objects involved are: bricks, paddle and ball.

## Discussion with stakeholders

I discussed the aspects of the game relating to the static game objects – the bricks with Lisa and Jay. The size, colour and number in a row and how many rows were decided along with the required behaviour of being visible or not. I explained that there will be a need to be able to easily check for an intersect between a brick and a moveable game object (the ball).

## Specify the proposed solution – Game Objects Detailed Requirements

1. There will need to be the ability to have more than one brick each with their own position
2. Each brick should be red in colour.
3. There should be the option to have the brick visible or invisible.
4. The brick will be in a particular position on the game screen as there will be a set of bricks.
5. There must be away of checking for an intersect between the ball and each individual brick.

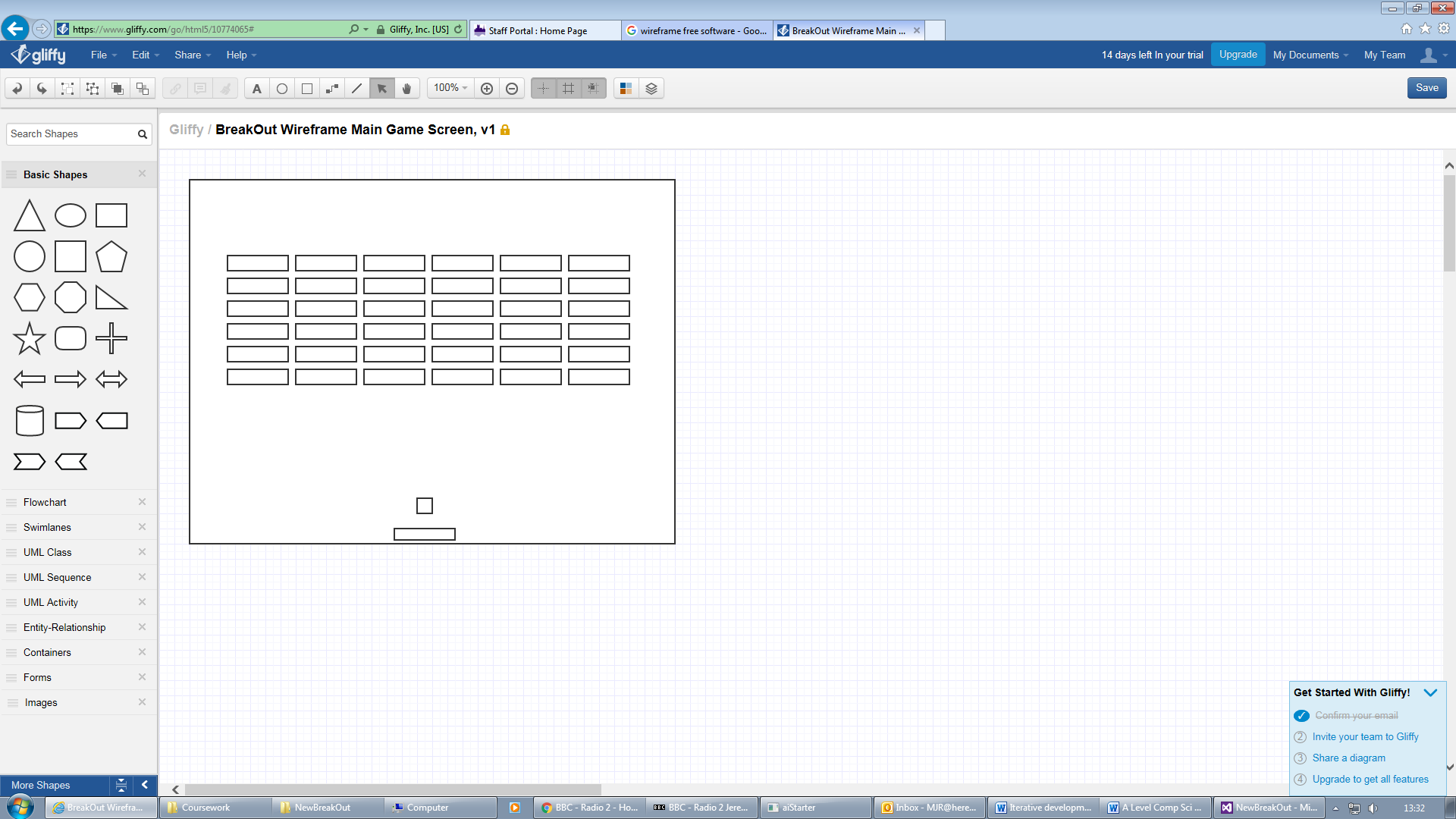
## **Decomposing the problem**

The diagram above shows the components of the solution to the problem. I will be creating a GameObject class which can be re-used for any static game objects.

Describe the Solution - Usability features

Below shows a wireframe of the main game screen. The main game objects are the paddle, ball and bricks. Colours will be as described in the requirements above. I’ve decided to go with a 6 x 6 block of bricks for the initial version of the game. It is possible extra levels maybe introduced using different brick layouts. However the stakeholders’ consider this to be an optional feature so this will be considered for implementation later on time permitting.

The users thought a help screen would be useful to explain the rules of the game and how to control the paddle. I plan to develop this in a later iteration once a working game has been put together as I may wish to use screenshots of actual game play in the help screen.  
  
***Extra requirement -*** There must be on-screen help available that explains the rules and how to play.



Describe the approach to testing – Usability Testing Plan  
I plan to get the stakeholders to test the game following post development system testing for function and robustness. They will play the game carrying out the usability tests described and record the result and any comments. I will also give them the opportunity to provide feedback on any desirable features they would like to see in any future versions.

|  |  |  |  |
| --- | --- | --- | --- |
| **Feature No.** | **Usability Test Description** | **Result** | **User Comments** |
| U1 | Are there a set of bricks with each individual brick being visible at start of game and becoming invisible after being hit by ball? |  |  |
| U2 | When all bricks have been hit by the ball does the game end showing it has been completed? |  |  |
| **Feedback on any extra features that would be appreciated in any future version of game** | | | |
|  | | | |

Describe the Solution  
I’ve decided to use Object-Oriented Programming to develop the solution. I’ve decided to create a GameObject class that implements static (non-moveable) game objects such as a bricks.

**Justification of choices made**

I’ve decided to use a Boolean data type for the blnIsVisible property as it will always be one of two values.

A rectangle object is the most sensible option for storing the position and size of the game objects. These are more efficient than for example modifying a PictureBox form control and should lead to better gameplay.

The CheckForIntersect method will allow a rectangle to be passed as a parameter and compared with the current game object to see if the two rectangles intersect at all.

|  |
| --- |
| **GameObject** |
| blnIsVisible:bool  recPosition:Rectangle  drawingBrushColour: System.Drawing.Brush |
| GameObject(int x, int y, int width, int height, bool blnIsVisible, Brush drawingBrushColour)  GetBlnIsVisible()  GetRecPosition()  GetDrawingBrushColour()  SetBlnIsVisible(isVisible:bool)  SetRecPosition(position:Rectangle)  SetDrawingBrushColour(brushColour:Brush)  MakeVisible()  MakeInvisible()  CheckForIntersect(recOther:Rectangle) |

Describe the approach to testing

During Development Testing Plan

I will test the GameObject class by writing unit tests to test the various methods. In the unit tests a new instance of GameObject will be created in order to carry out the tests. The tests are detailed in the table below.  
  
**Note on test data**All the following tests are based on the following GameObject. Object name: brick, X = 41, Y = 100, width = 80, height = 20, isVisible = True, Brushes.Red.

|  |  |  |  |
| --- | --- | --- | --- |
| **Test No.** | **Description of Test** | **Test Data** | **Expected Result** |
| DD1.1 | **TestGetBlnIsVisible**  Check GetBlnIsVisible() returns True | call to GetBlnIsVisible() | True is returned |
| DD1.2 | **TestPosition** Check Position returns a Rectangle with X = 41, Y = 100, width = 80, height = 20 | Call to GetRecPosition() | Rectangle is returned with values X = 41, Y = 100, width = 80, height = 20 |
| DD1.3 | **TestBrushColour**  Check the get and set methods for BrushColour() | Call to GetDrawingBrushColour() Call to SetDrawingBrushColour(Brushes.Yellow)  Call to GetDrawingBrushColour() | Get should return Brushes.Red  Set Brushes.Yellow  Check Get returns Brushes.Yellow |
| DD1.4 | **TestVisibility**  Check whether methods to change GameObject BlnIsVisible property work | Call to MakeInvisible() Call to GetBlnIsVisible()  Call to MakeVisible() | Changes BlnIsVisible property to False (call to GetBlnIsVisible() should return False).  Changes BlnIsVisible property to True (call to GetBlnIsVisible() should return True) |
| DD1.5 | **TestCheckForIntersect**  Create a second brick with the same coordinates as brick one and test the brick one CheckForIntersect method against the other brick’s rectangle | Call to CheckForIntersect(brickOther.GetRecPosition()) | True is returned |
| DD1.6 | **TestFailedCheckForIntersect**  Create another brick with different coordinates to brick one but same size (X = 0, Y = 0, width = 80, height = 20) and test the brick one CheckForIntersect method against the other brick’s rectangle | Call to CheckForIntersect(brickOther.GetRecPosition()) | Returns False |
| DD1.7 | **TestBorderlineCheckForIntersect**  Create another brick with coordinates just overlapping brick one but same size (X = 120, Y = 119, width = 80, height = 20) and test the brick one CheckForIntersect method against the other brick’s rectangle | Call to CheckForIntersect(brickOther.GetRecPosition()) | True is returned |

### Post Development System Testing for Function and Robustness Plan

|  |  |  |  |
| --- | --- | --- | --- |
| **Test No.** | **Description of Test** | **Test Data** | **Expected Result** |
| PD1 | Bricks appears correctly on screen at start of game | Click start game | Game starts with 6x6 red bricks each 80 width x20 height in pixels arranged uniformly on-screen |
| PD2 | Bricks disappear when hit by ball | Direct ball into middle of brick | Brick disappears |
| PD3 | Bricks disappear when hit by ball | Direct ball across corner of brick | Brick disappears |
| PD4 | When last brick is hit the game is complete. | Direct ball into last brick | Game ends |
| PD5 | Bricks remain visible on screen when ball does not hit them | Direct ball away from a set of bricks in different angles | Bricks remain visible |
| PD6 | Bricks remain visible on screen when ball does not hit them | Direct ball just missing a brick (by a very small amount) | Bricks remain visible |

## Describe the solution – Pseudo code algorithms and justification

Public Class GameObject

bool blnIsVisible   
Rectangle recPosition

**Justification of how this forms part of the complete solution**

|  |  |
| --- | --- |
| **Game Object Detailed Requirement Number** | Comments on how algorithms meet requirements |
| 11 | Brick objects can be created from this class. Could be stored in an array. |
| 12 | The GameObject class allows objects to have an associated colour including Red |
| 13 | Yes there is the option have brick invisible or visible |
| 14 | The position property is of class rectangle which allows for X and Y coordinates on screen as well as height and width. |
| 15 | The class has a checkForIntersect method to check the current object against any other rectangle for an intersect. |

System.Drawing.Brush drawingBrushColour

GameObject(x,y,width,height,blnIsVisible,drawingBrushColour)  
 this.blnIsVisible = blnIsVisible  
 recPosition = New Rectangle(x,y,width,height)

this.drawingBrushColour = drawingBrushColour  
End Constructor

Function GetBlnIsVisible()

Return blnIsVisible

End Function

Function GetRecPosition()  
 Return recPosition

End Function

Function GetDrawingBrushColour()

Return drawingBrushColour

End Function

Procedure SetBlnIsVisible(isVisible)  
 blnIsVisible = isVisible

End Procedure

Procedure SetRecPosition(position)  
 recPosition = position

End Procedure

Procedure SetDrawingBrushColour(brushColour)

drawingBrushColour = brushColour

End Procedure

Procedure MakeVisible()  
 blnIsVisible = true

End Procedure  
  
Procedure MakeInVisible()  
 blnIsVisible = false

End Procedure  
  
Function CheckForIntersect(recOther)

If recPosition intersects with recOther Then  
 Return true  
 Else

Return false  
 End If

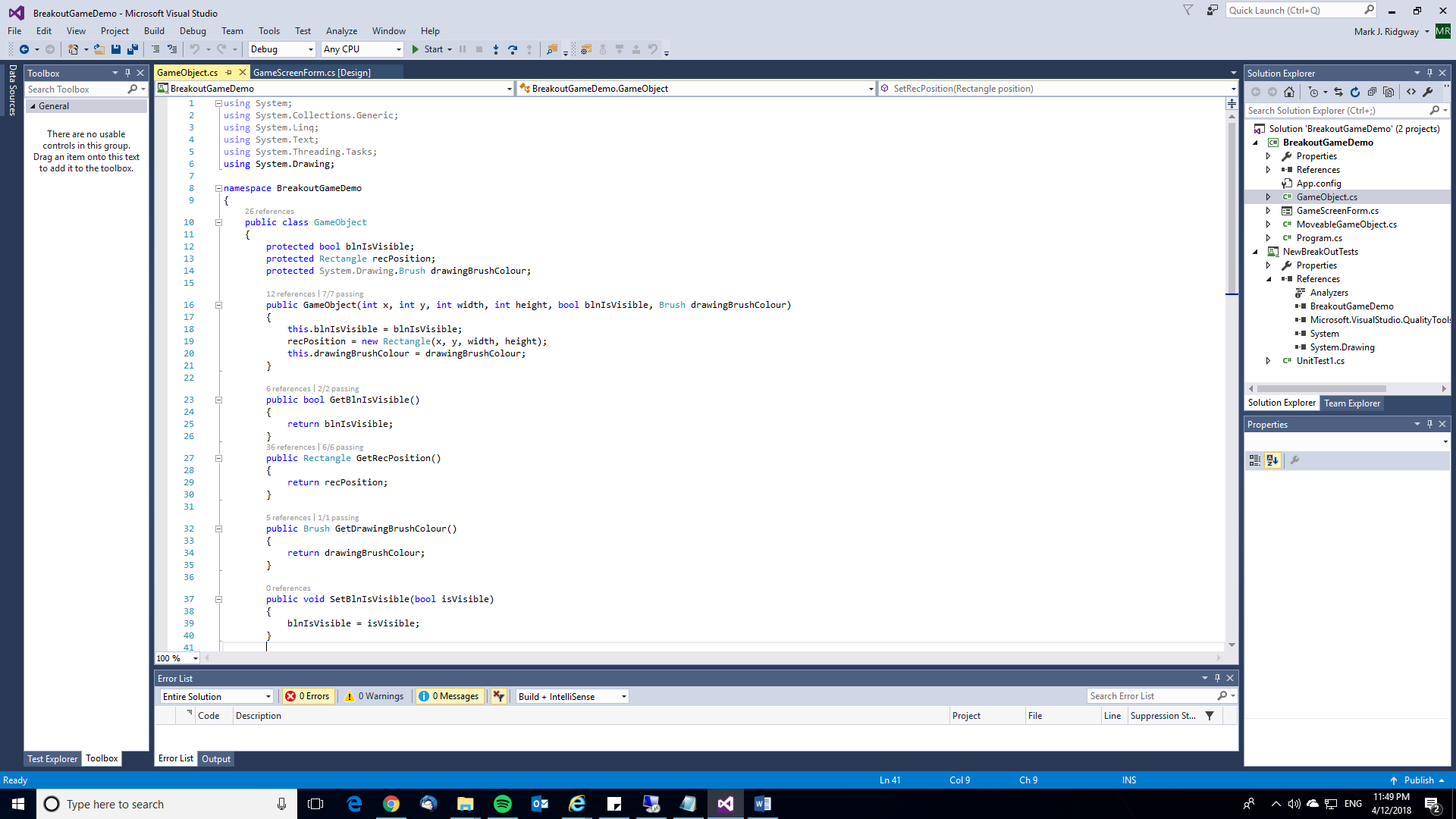
End Function

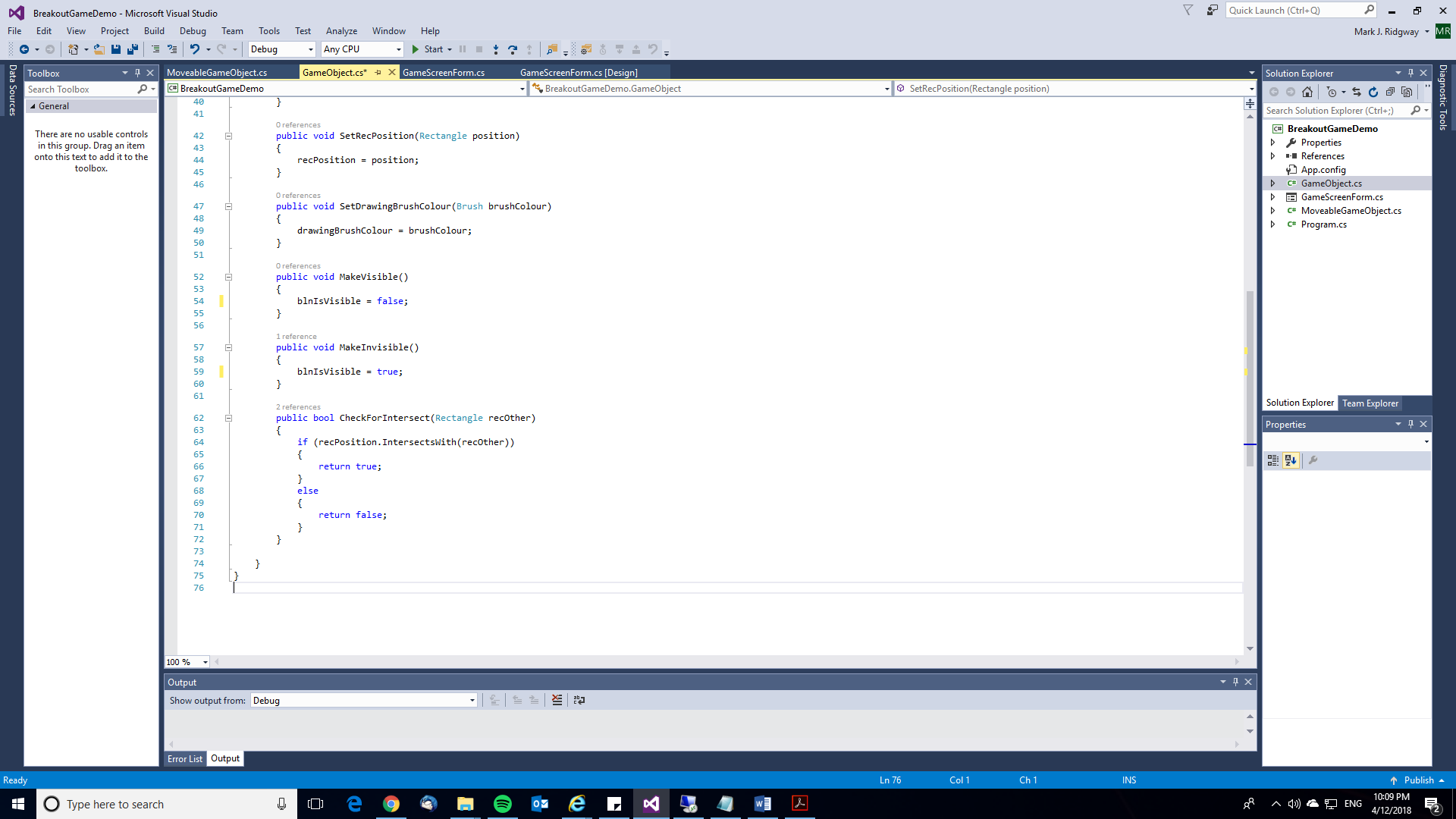
End Class

## Developing the solution – Coding

Note: you will be expected to annotate/in-line comment any sections of code that are more complex and need explaining. There isn’t much to explain apart from possibly the checkForIntersect method in this class. You aren’t expected to comment every line of code for the sake of it when you have used sensible identifiers and code is understandable without. In C# we write a comment like this:  
// This is a single-line comment note the special symbols at the beginning.

/\*This is a  
multiline comment\*/





## Testing to inform development

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Test No.** | **Description of Test** | **Test Data** | **Expected Result** | **As expected? Comment if necessary** |
| DD1.1 | **TestGetBlnIsVisible**  Check GetBlnIsVisible() returns True | call to GetBlnIsVisible() | True is returned | Y |
| DD1.2 | **TestPosition** Check Position returns a Rectangle with X = 41, Y = 100, width = 80, height = 20 | Call to GetRecPosition() | Rectangle is returned with values X = 41, Y = 100, width = 80, height = 20 | Y |
| DD1.3 | **TestBrushColour**  Check the get and set methods for BrushColour() | Call to GetDrawingBrushColour() Call to SetDrawingBrushColour(Brushes.Yellow)  Call to GetDrawingBrushColour() | Get should return Brushes.Red  Set Brushes.Yellow  Check Get returns Brushes.Yellow | Y |
| DD1.4 | **TestVisibility**  Check whether methods to change GameObject BlnIsVisible property work | Call to MakeInvisible() Call to GetBlnIsVisible()  Call to MakeVisible() | Changes BlnIsVisible property to False (call to GetBlnIsVisible() should return False).  Changes BlnIsVisible property to True (call to GetBlnIsVisible() should return True) | N – test fails. See evidence of remedial action below |
| DD1.5 | **TestCheckForIntersect**  Create a second brick with the same coordinates as brick one and test the brick one CheckForIntersect method against the other brick’s rectangle | Call to CheckForIntersect(brickOther.GetRecPosition()) | True is returned | Y |
| DD1.6 | **TestFailedCheckForIntersect**  Create another brick with different coordinates to brick one but same size (X = 0, Y = 0, width = 80, height = 20) and test the brick one CheckForIntersect method against the other brick’s rectangle | Call to CheckForIntersect(brickOther.GetRecPosition()) | Returns False | Y |
| DD1.7 | **TestBorderlineCheckForIntersect**  Create another brick with coordinates just overlapping brick one but same size (X = 120, Y = 119, width = 80, height = 20) and test the brick one CheckForIntersect method against the other brick’s rectangle | Call to CheckForIntersect(brickOther.GetRecPosition()) | True is returned | Y |

***Evidence of during development testing***

***Unit Test Code*** *UnitTest1.cs*

using System;

using Microsoft.VisualStudio.TestTools.UnitTesting;

using BreakoutGameDemo;

using System.Drawing;

namespace NewBreakOutTests

{

[TestClass]

public class UnitTest1

{

[TestMethod]

public void TestGetBlnIsVisible()

{

GameObject brick = new GameObject(41, 100, 80, 20, true, Brushes.Red);

Assert.AreEqual(true, brick.GetBlnIsVisible());

}

[TestMethod]

public void TestPosition()

{

GameObject brick = new GameObject(41, 100, 80, 20, true, Brushes.Red);

Rectangle rec = brick.GetRecPosition();

Rectangle testRec = new Rectangle(41, 100, 80, 20);

Assert.AreEqual(rec, testRec);

}

[TestMethod]

public void TestBrushColour()

{

GameObject brick = new GameObject(41, 100, 80, 20, true, Brushes.Red);

Assert.AreEqual(System.Drawing.Brushes.Red, brick.GetDrawingBrushColour());

brick.SetDrawingBrushColour(System.Drawing.Brushes.Yellow);

Assert.AreEqual(System.Drawing.Brushes.Yellow, brick.GetDrawingBrushColour());

}

[TestMethod]

public void TestVisibility()

{

GameObject brick = new GameObject(41, 100, 80, 20, true, Brushes.Red);

brick.MakeInvisible();

Assert.AreEqual(false, brick.GetBlnIsVisible());

brick.MakeVisible();

Assert.AreEqual(true, brick.GetBlnIsVisible());

}

[TestMethod]

public void TestCheckForIntersect()

{

GameObject brick = new GameObject(41, 100, 80, 20, true, Brushes.Red);

GameObject brickOther = new GameObject(41, 100, 80, 20, true, Brushes.Red);

Assert.AreEqual(true, brick.CheckForIntersect(brickOther.GetRecPosition()));

}

[TestMethod]

public void TestFailedCheckForIntersect()

{

GameObject brick = new GameObject(41, 100, 80, 20, true, Brushes.Red);

GameObject brickOther = new GameObject(0, 0, 80, 20, true, Brushes.Red);

Assert.AreEqual(false, brick.CheckForIntersect(brickOther.GetRecPosition()));

}

[TestMethod]

public void TestBorderlineCheckForIntersect()

{

GameObject brick = new GameObject(41, 100, 80, 20, true, Brushes.Red);

GameObject brickOther = new GameObject(120, 119, 80, 20, true, Brushes.Red);

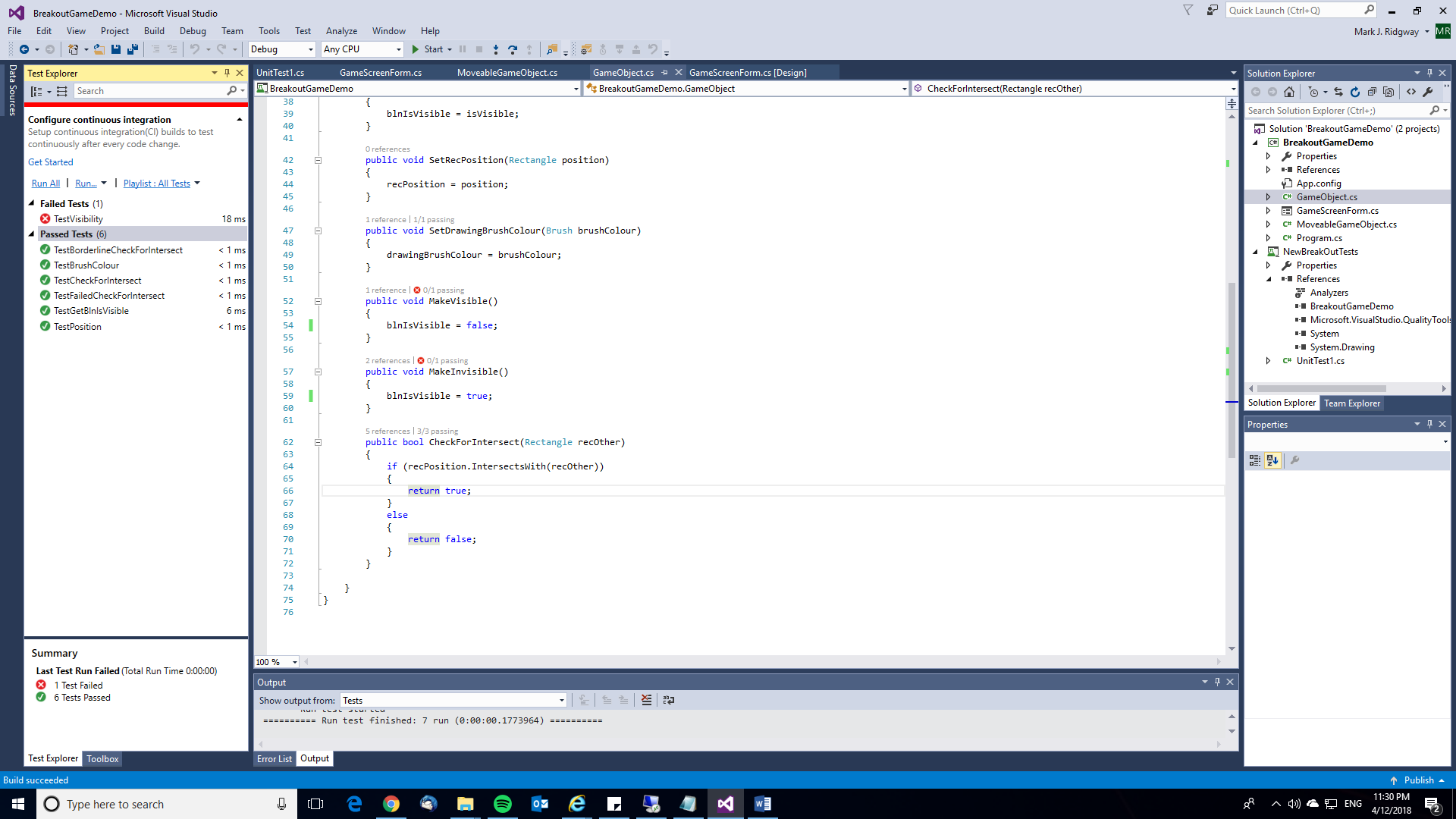
Assert.AreEqual(true, brick.CheckForIntersect(brickOther.GetRecPosition()));

}

}

}

***Proof of Test Results including remedial actions and justification if any required***



Evidence of remedial action for DD1.4 and justification

Initially the MakeVisible and MakeInVisible methods were coded as follows:

public void MakeVisible()

{

blnIsVisible = false;

}

public void MakeInvisible()

{

blnIsVisible = true;

}

This is clearly a logic error. MakeVisible should set blnIsVisible to True (not False as it is) and vice versa for MakeInVisible. Here is the corrected code below and proof the test passes.

public void MakeVisible()

{

blnIsVisible = true;

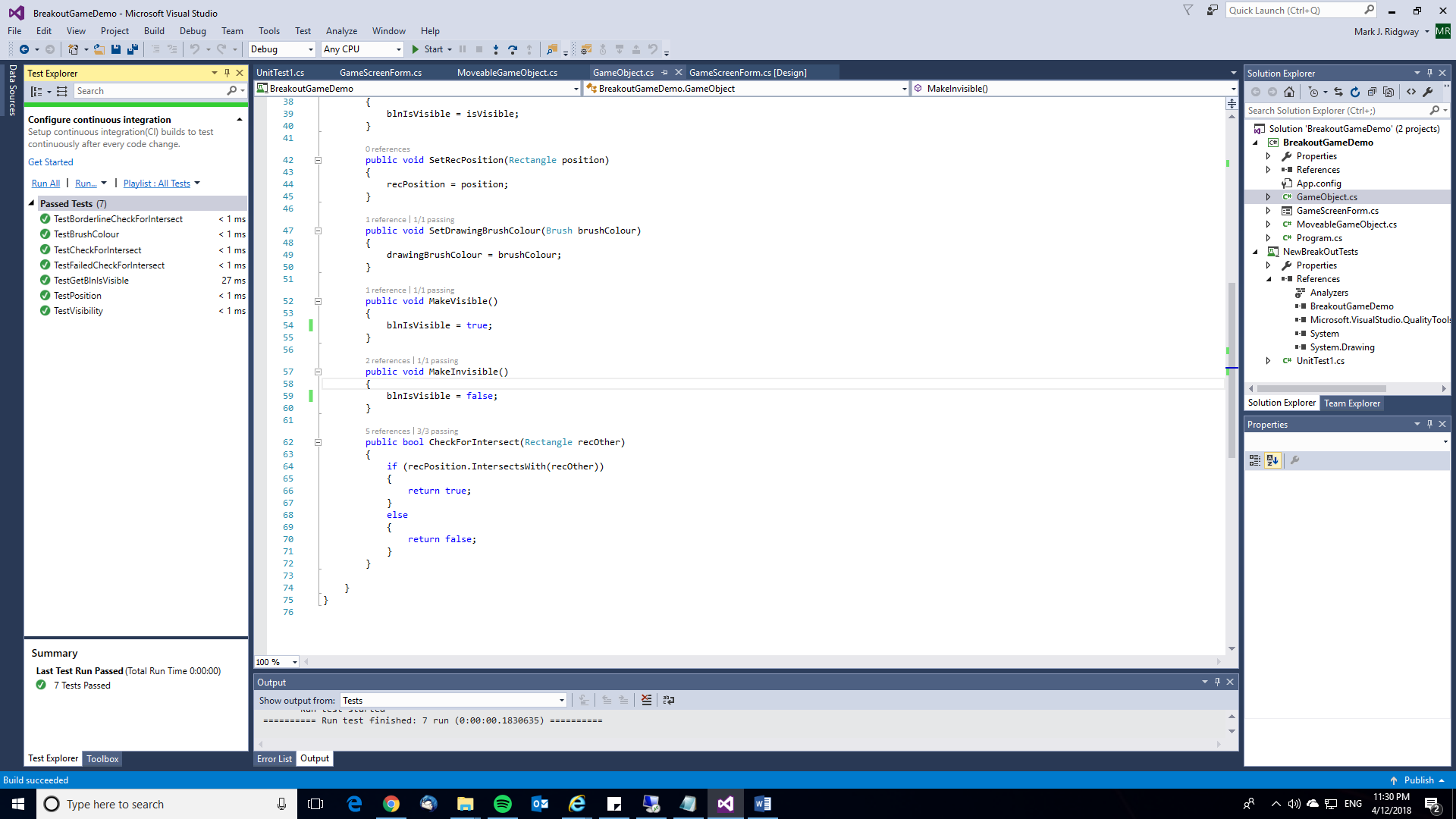
}

public void MakeInvisible()

{

blnIsVisible = false;

}



## Stakeholder review and sign-off of iteration

The stakeholders were happy with the progress made this iteration and the work completed so far. They have allowed me to move on to the next iteration. They mentioned they’d like the ball to start actually on the paddle not just above as it is shown in the wireframe. I will consider this when working on the next iteration.

**END OF ITERATION 1**

# Iteration 2 – Moveable Game Object

## Discussion with Stakeholders

The design aspects of the moveable game objects were discussed with Lisa and Jay. This included the design (colour, shape and size) and behaviour of these objects. In the end it was decided that the ball should be the same colour as the paddle and the paddle should only move left and right (not up and down).

## Specify the proposed solution – Moveable Game Objects Detailed Requirements

1. There must be a paddle which can be moved left and right using the mouse.
2. The paddle should be pink in colour and 80 pixels wide and 7 high.
3. The initial position of the paddle should be middle-bottom of screen.
4. The paddle should only move left and right within the bounds of the screen.
5. The ball should be 16 x 16 pixels and yellow in colour.
6. The initial position of the ball should be centred on the paddle.
7. The ball should be glued at the start of the game until launched and also at the end when the last brick is hit or lives lost.
8. The ball should be unglued throughout game play (after launch).
9. The ball can move in any direction on the game screen (left, right and up and down).
10. The ball will have an associated speed and X and Y velocity which affects movement.

## Decompose the problem

I’ve added to the structure diagram to show the components of the solution to the ball and paddle (moveable game objects) and the game screen itself.

## Describe the Solution - Usability features

When the mouse moves across the window the paddle will move into position according to the X coordinate of the mouse pointer. The stakeholders have preference for use of mouse rather than having to remember keys to press as they feel it will make for a more intuitive interface I happen to agree. In discussions with the stakeholders I identified another possible usability feature and that is the option to change the speed at which the paddle moves. I plan to design my solution so this feature can easily be implemented if desired.

## Describe the approach to testing – Usability Testing Plan

|  |  |  |  |
| --- | --- | --- | --- |
| **Feature No.** | **Usability Test Description** | **Result** | **User Comments** |
| U3 | Does the paddle move left and right within the bounds of the screen with the mouse as you’d expect? |  |  |
| U4 | Is the design and positioning of the paddle correct? (A pink 80x7 pixel paddle with the initial position of middle-bottom was specified) |  |  |
| U5 | Are the design, size and initial positioning of the ball as you expect? (A yellow ball of 16x16 pixels with an initial position of centre-paddle was specified). |  |  |
| U6 | Does the behaviour of the ball seem sensible? (Ball should be unglued throughout game play after launch, it can move in any direction and has a speed and X and Y velocity). |  |  |
| **Feedback on any extra features that would be appreciated in any future version of game** | | | |
|  | | | |

## Describe the Solution

I’m going to use inheritance to customize the generic GameObject class to create a MoveableGameObject class that implements moveable game objects such as a paddle and ball. I’m going to need to change the position of the paddle and ball in different ways (the paddle can only have its X coordinate changed whereas the ball will usually have both X and Y changed). I can get around this by having two changePosition methods (called the same) one that takes in one parameter X and one that takes in two parameters X and Y.

|  |
| --- |
| **GameObject** |
| blnIsVisible:bool  recPosition:Rectangle  drawingBrushColour: System.Drawing.Brush |
| GameObject(int x, int y, int width, int height, bool blnIsVisible, Brush drawingBrushColour)  GetBlnIsVisible()  GetRecPosition()  GetDrawingBrushColour()  SetBlnIsVisible(isVisible:bool)  SetRecPosition(position:Rectangle)  SetDrawingBrushColour(brushColour:Brush)  MakeVisible()  MakeInvisible()  CheckForIntersect(recOther:Rectangle) |

**Justification of choices made for MoveableGameObject**

I’ve decided to use a Boolean data type for the blnIsGlued property as it will always be one of two values.

sngSpeed, sngXVel and sngYVel are all float data types as these allow fractional numbers to be stored allowing more finite control over speed and velocity.

There are two changePosition methods one which allows both the horizontal and vertical (X and Y coordinates) to be changed and the other which only allows the horizontal (X coordinate) position to be changed. The paddle will only require the X coordinate to be changed whereas the ball will need both changed.

|  |
| --- |
| **MoveableGameObject** |
| fltSpeed,fltXVel,fltYVel : float  blnIsGlued : bool |
| MoveableGameObject(int x, int y, int width, int height, bool blnIsVisible, float fltSpeed, float fltXVel, float fltYVel, bool blnIsGlued, Brush drawingBrushColour)  getFltSpeed()  getfltXVel()  getfltYVel()  getBlnIsGlued()  setFltSpeed(float speed)  setFltXVel(float xVel)  setFltYVel(float yVel)  Glue()  UnGlue()  ChangePosition(int x, int y)  ChangePosition(int x) |

## Describe the approach to testing

During Development Testing Plan

I will test the MoveableGameObject class by writing unit tests to test the various methods. In the unit tests a new instance of MoveableGameObject will be created in order to carry out the tests. The tests are detailed in the table below.  
  
**Note on test data**All the following tests are based on the following MoveableGameObject. Object name: ball, X = 298, Y = 414, width = 20, height = 20, isVisible = True, speed = 20, XVel = 8.161641, YVel = 8.161641, isGlued = True, Brushes.Yellow.

|  |  |  |  |
| --- | --- | --- | --- |
| **Test No.** | **Description of Test** | **Test Data** | **Expected Result** |
| DD2.1 | Check getBlnIsGlued() returns True | call to getBlnIsGlued() | True is returned |
| DD2.2 | Check GetRecPosition returns a Rectangle with X = 298, Y = 414, width = 20, height = 20 | Call to GetRecPosition() | Rectangle is returned with values X = 298, Y = 414, width = 20, height = 20 |
| DD2.3 | Check whether methods to change GameObject blnIsGlued property work | Call to UnGlue() Call to IsGlued()  Call to Glue()  Call to IsGlued() | Changes blnIsGlued property to False (call to IsGlued() should return False).  Changes blnIsGlued property to True (call to IsGlued() should return True) |
| DD2.4 | Check that methods to change the objects position work | Call to ChangePosition(100,100)  Call to GetRecPosition()  Call to ChangePosition(200)  Call to GetRecPosition() | Rectangle is returned with X = 100, Y = 100.  Rectangle is returned with X = 200, Y = 100. |

### Post Development System Testing for Function and Robustness Plan

|  |  |  |  |
| --- | --- | --- | --- |
| **Test No.** | **Description of Test** | **Test Data** | **Expected Result** |
| PD1 | Ball appears glued on the middle of the paddle at start of game. Paddle is positioned middle-bottom initially. | Click start game | Game starts with 20x20 pixel ball (yellow) glued to the middle of the paddle. 80x7 pixel paddle (pink) is positioned middle-bottom initially (but moves with mouse). |
| PD2 | Ball stays without bounds of screen except when missing the paddle and falling through the bottom of the game screen. | Using mouse to control paddle bounce ball against all sides of game screen and allow ball to miss paddle. | Ball bounces off left, right and top walls but falls past bottom of game screen (and life is lost). |
| PD3 | Check paddle can only move left and right (not up and down). Ensure paddle stays within bounds of screen. | Move mouse in all directions to ensure paddle is only moving left and right. Attempt to move paddle past left and right edge of game screen. | Paddle moves left and right in response to mouse movement and won’t go beyond the bounds of the screen. The paddle will not move up and down. |
| PD4 | Check ball is released in response to launch | Click mouse at start of game | Ball should be launched from paddle and move upwards. |
| PD5 | Check ball is glued after last life lost or last brick hit. | Play game and lose all lives. Play game and hit all bricks. | Ball is glued on loss of last life or last brick being hit. |
| PD6 | Check ball can move in any direction on the game screen. | Play game using mouse to bounce ball and check it can move in all directions. | Ball able to move in all directions. |

## 

## Describe the solution – Pseudo code algorithms and justification

Public Class MoveableGameObject

Inherits GameObject

bool blnIsGlued

float fltSpeed, fltXVel, fltYVel  
 MoveableGameObject(x,y,width,height,blnIsVisible,fltSpeed,fltXVel,fltYVel,blnIsGlued,drawingBrushColour)

: base(x,y,width,height,blnIsVisible,drawingBrushColour)

this.fltSpeed = fltSpeed;

this.fltXVel = fltXVel;

this.fltYVel = fltYVel;

this.blnIsGlued = blnIsGlued;

End Constructor

Function GetFltSpeed()

Return fltSpeed

End Function

Function GetFltXVel()

Return fltXVel

End Function

Function GetFltYVel()

Return fltYVel

End Function

Function GetBlnIsGlued()

Return blnIsGlued

End Function

Procedure SetFltSpeed(speed)

fltSpeed = speed

End Procedure

Procedure SetFltXVel(xVel)

fltXVel = xVel

End Procedure

Procedure SetFltYVel(yVel)

fltYVel = yVel

End Procedure

Procedure Glue()

blnIsGlued = True

End Procedure

Procedure UnGlue()

blnIsGlued = False

End Procedure

Procedure ChangePosition(x, y)

recPosition.Location = new Point(x, y)

End Procedure

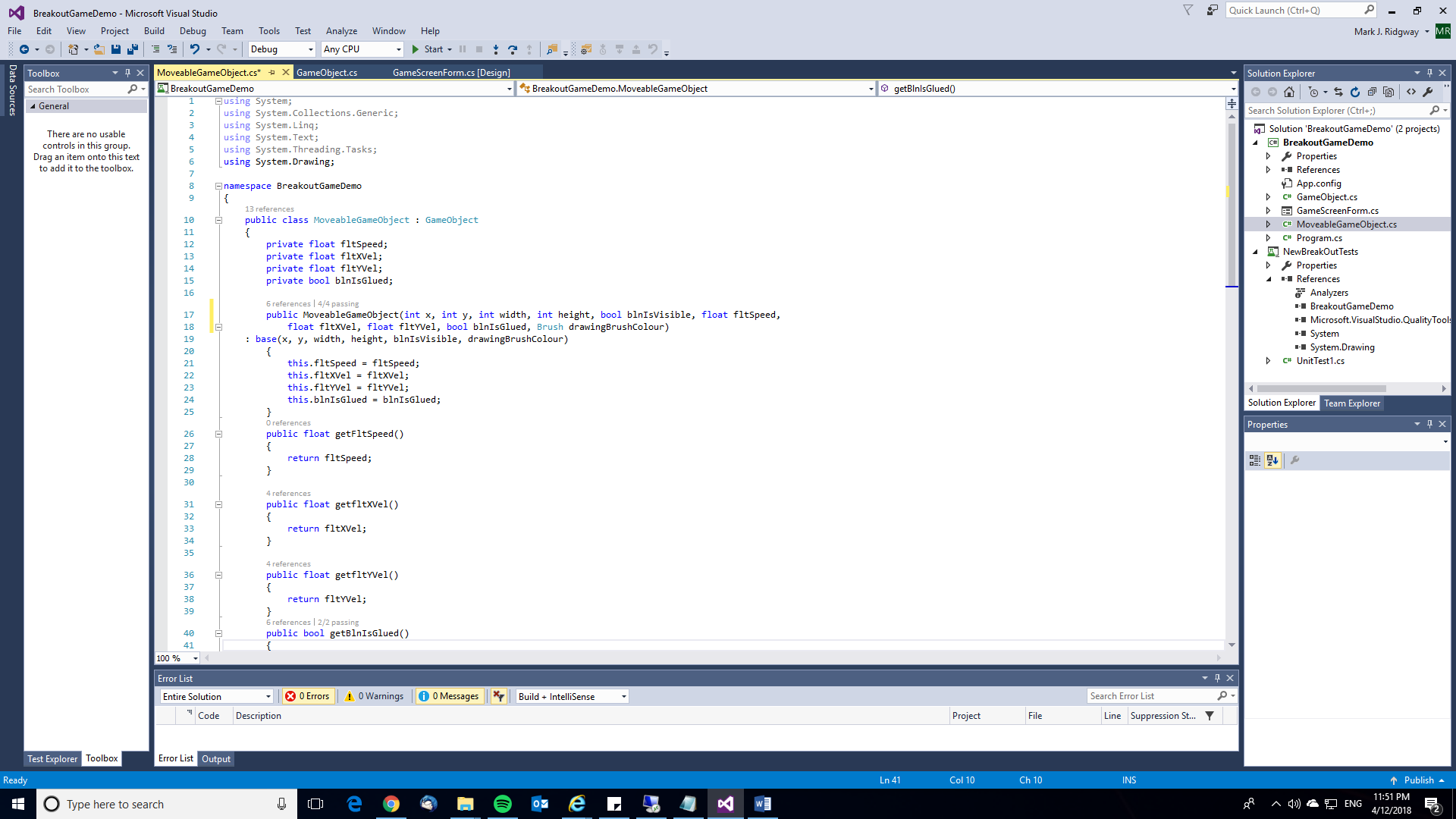
Procedure ChangePosition(x)

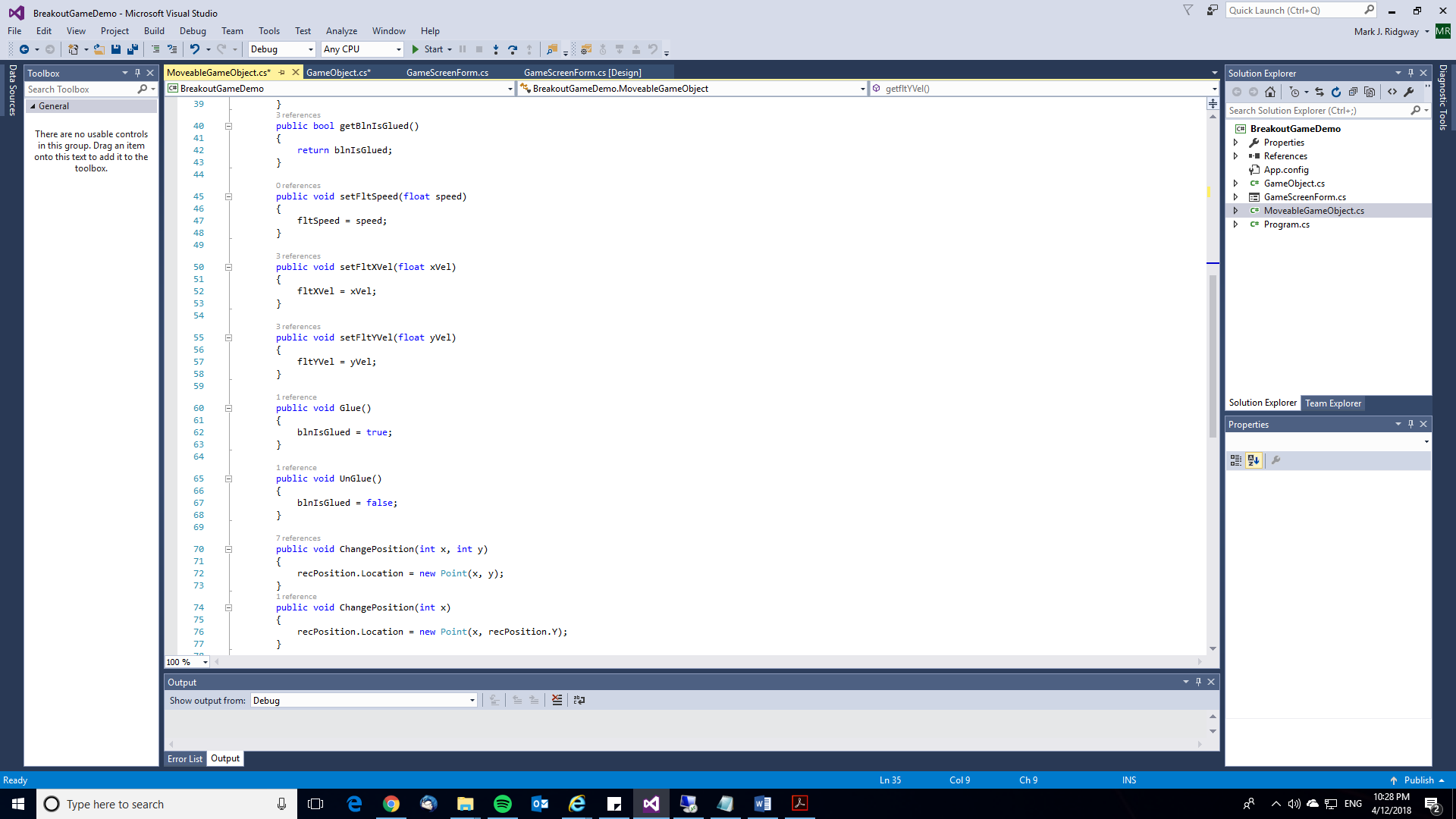
recPosition.Location = new Point(x, Position().Y)

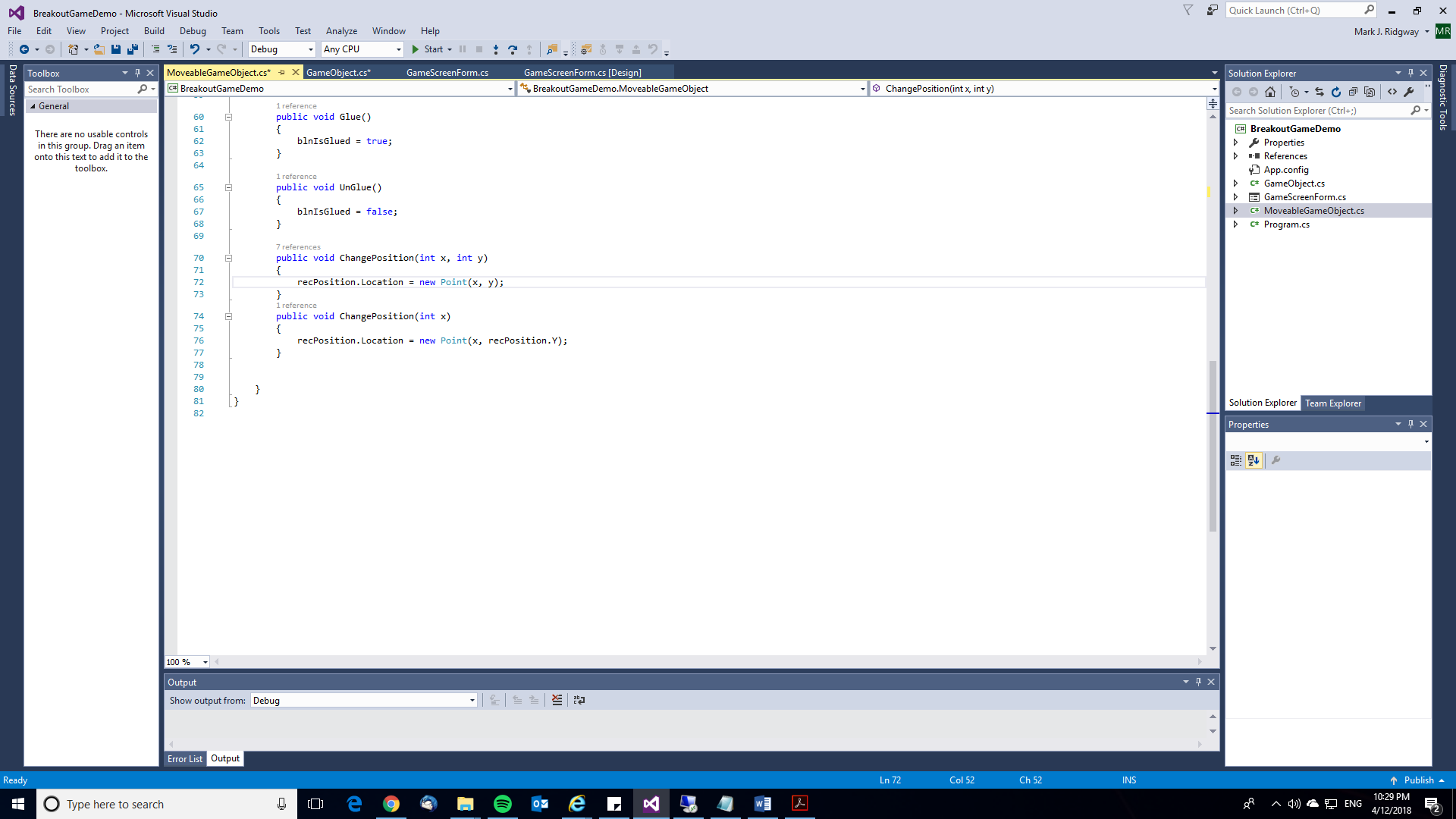
End Procedure

End Class

## Developing the solution – Coding







## Testing to inform development

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Test No.** | **Description of Test** | **Test Data** | **Expected Result** | **As expected? Comment if necessary** |
| DD2.1 | Check getBlnIsGlued() returns True | call to getBlnIsGlued() | True is returned | Y |
| DD2.2 | Check GetRecPosition returns a Rectangle with X = 298, Y = 414, width = 20, height = 20 | Call to GetRecPosition() | Rectangle is returned with values X = 298, Y = 414, width = 20, height = 20 | Y |
| DD2.3 | Check whether methods to change GameObject blnIsGlued property work | Call to UnGlue() Call to IsGlued()  Call to Glue()  Call to IsGlued() | Changes blnIsGlued property to False (call to IsGlued() should return False).  Changes blnIsGlued property to True (call to IsGlued() should return True) | Y |
| DD2.4 | Check that methods to change the objects position work | Call to ChangePosition(100,100)  Call to GetRecPosition()  Call to ChangePosition(200)  Call to GetRecPosition() | Rectangle is returned with X = 100, Y = 100.  Rectangle is returned with X = 200, Y = 100. | Y |

***Evidence of during development testing***

***Unit Test Code*** *UnitTest1.cs*

[TestClass]

public class MoveableGameObjectTests

{

[TestMethod]

public void TestGetBlnIsGlued()

{

MoveableGameObject ball = new MoveableGameObject(298, 414, 20, 20, true, 20, 8.161641f, 8.161641f, true, System.Drawing.Brushes.Yellow);

Assert.AreEqual(true, ball.getBlnIsGlued());

}

[TestMethod]

public void TestPosition()

{

MoveableGameObject ball = new MoveableGameObject(298, 414, 20, 20, true, 20, 8.161641f, 8.161641f, true, System.Drawing.Brushes.Yellow);

Rectangle rec = ball.GetRecPosition();

Rectangle testRec = new Rectangle(298,414,20,20);

Assert.AreEqual(rec, testRec);

}

[TestMethod]

public void TestGlueing()

{

MoveableGameObject ball = new MoveableGameObject(298, 414, 20, 20, true, 20, 8.161641f, 8.161641f, true, System.Drawing.Brushes.Yellow);

ball.UnGlue();

Assert.AreEqual(false, ball.getBlnIsGlued());

ball.Glue();

Assert.AreEqual(true, ball.getBlnIsGlued());

}

[TestMethod]

public void TestChangePosition()

{

MoveableGameObject ball = new MoveableGameObject(298, 414, 20, 20, true, 20, 8.161641f, 8.161641f, true, System.Drawing.Brushes.Yellow);

Rectangle testRecOne = new Rectangle(100, 100, 20, 20);

ball.ChangePosition(100, 100);

Assert.AreEqual(ball.GetRecPosition(), testRecOne);

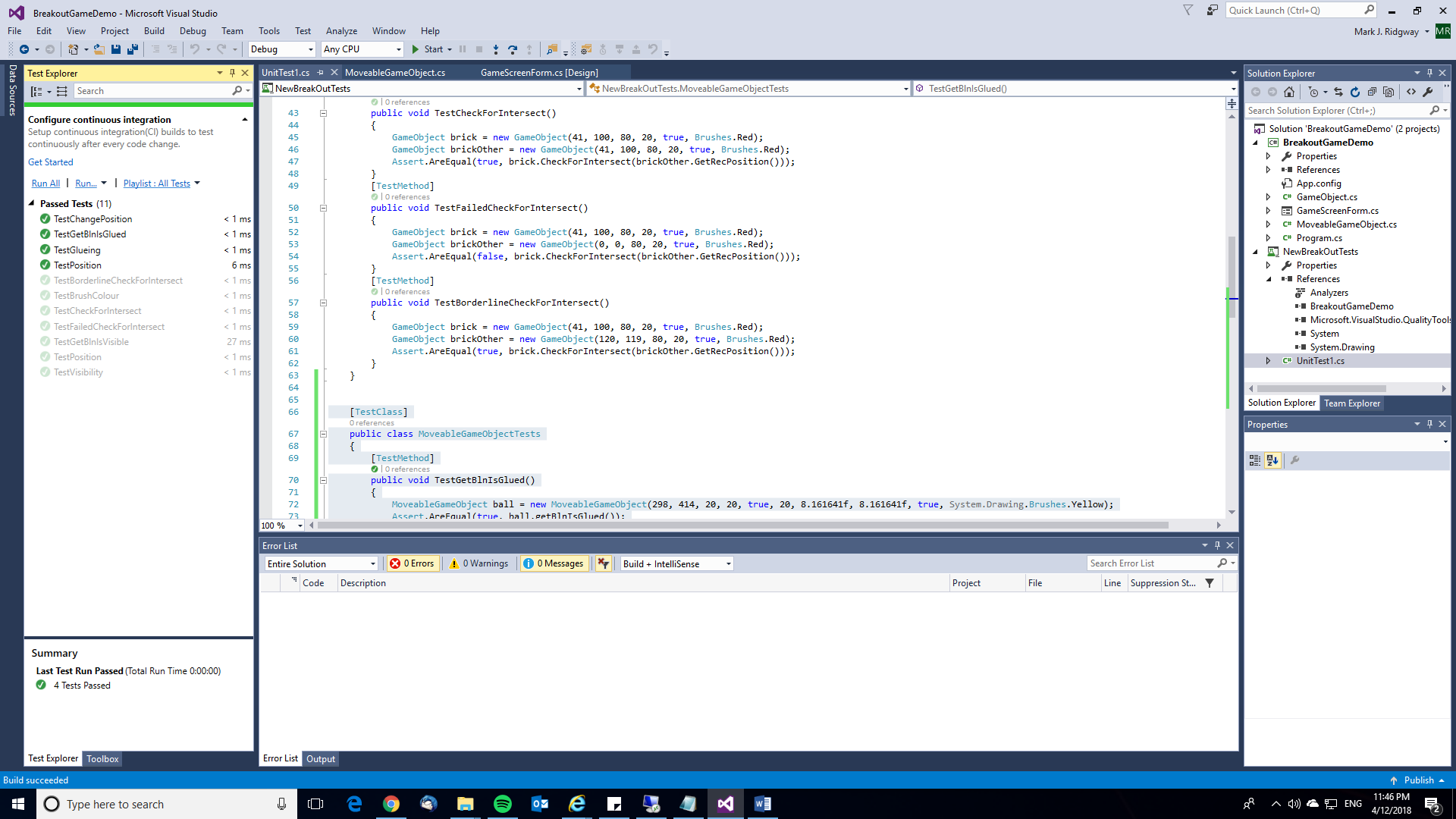
ball.ChangePosition(200);

Rectangle testRecTwo = new Rectangle(200, 100, 20, 20);

Assert.AreEqual(ball.GetRecPosition(), testRecTwo);

}

}

***Proof of Test Results***

## Stakeholder review and sign-off of iteration

The stakeholders were happy with the progress made this iteration and the work completed so far. They have allowed me to move on to the next iteration.

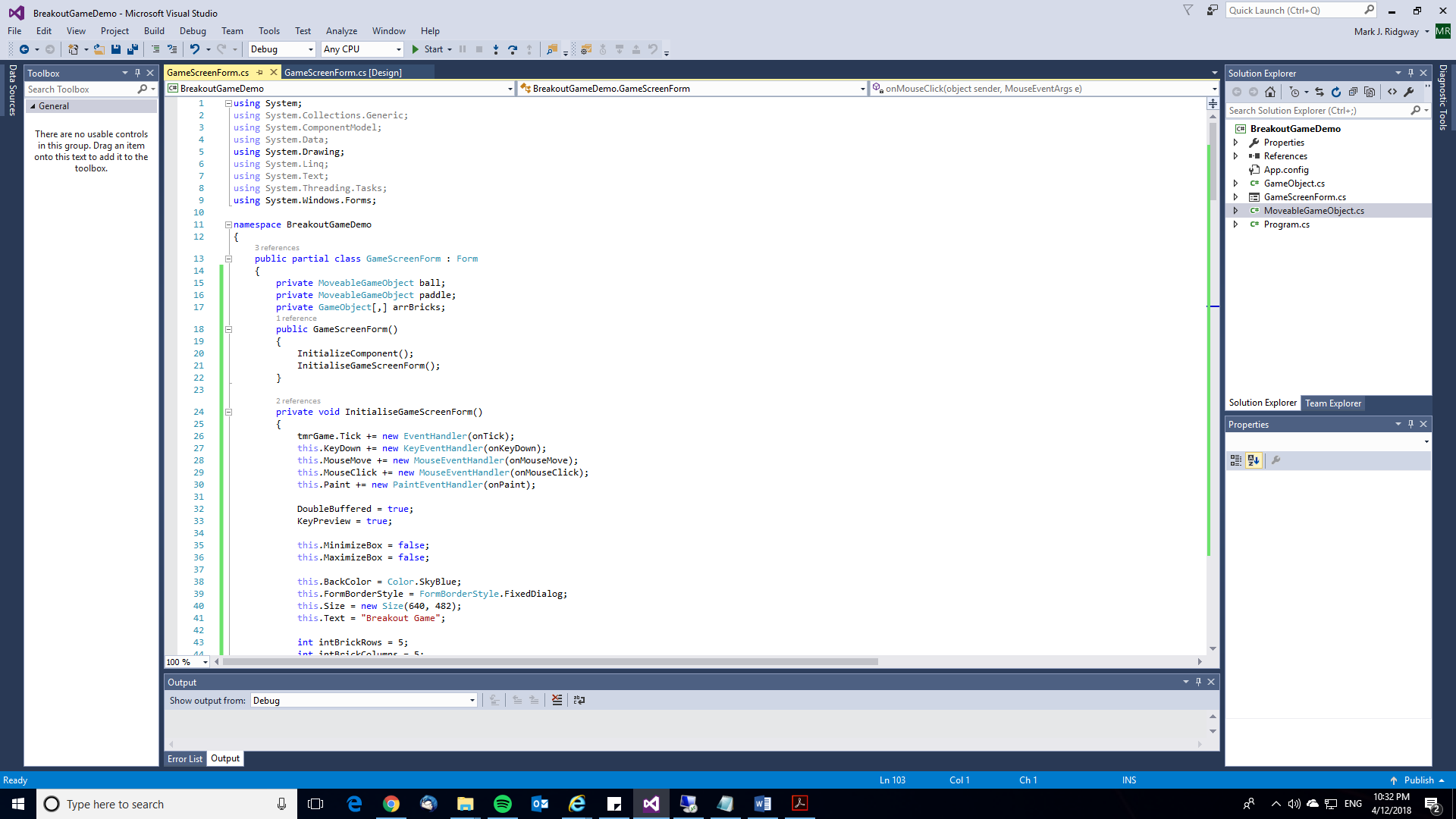
**END OF ITERATION 2**

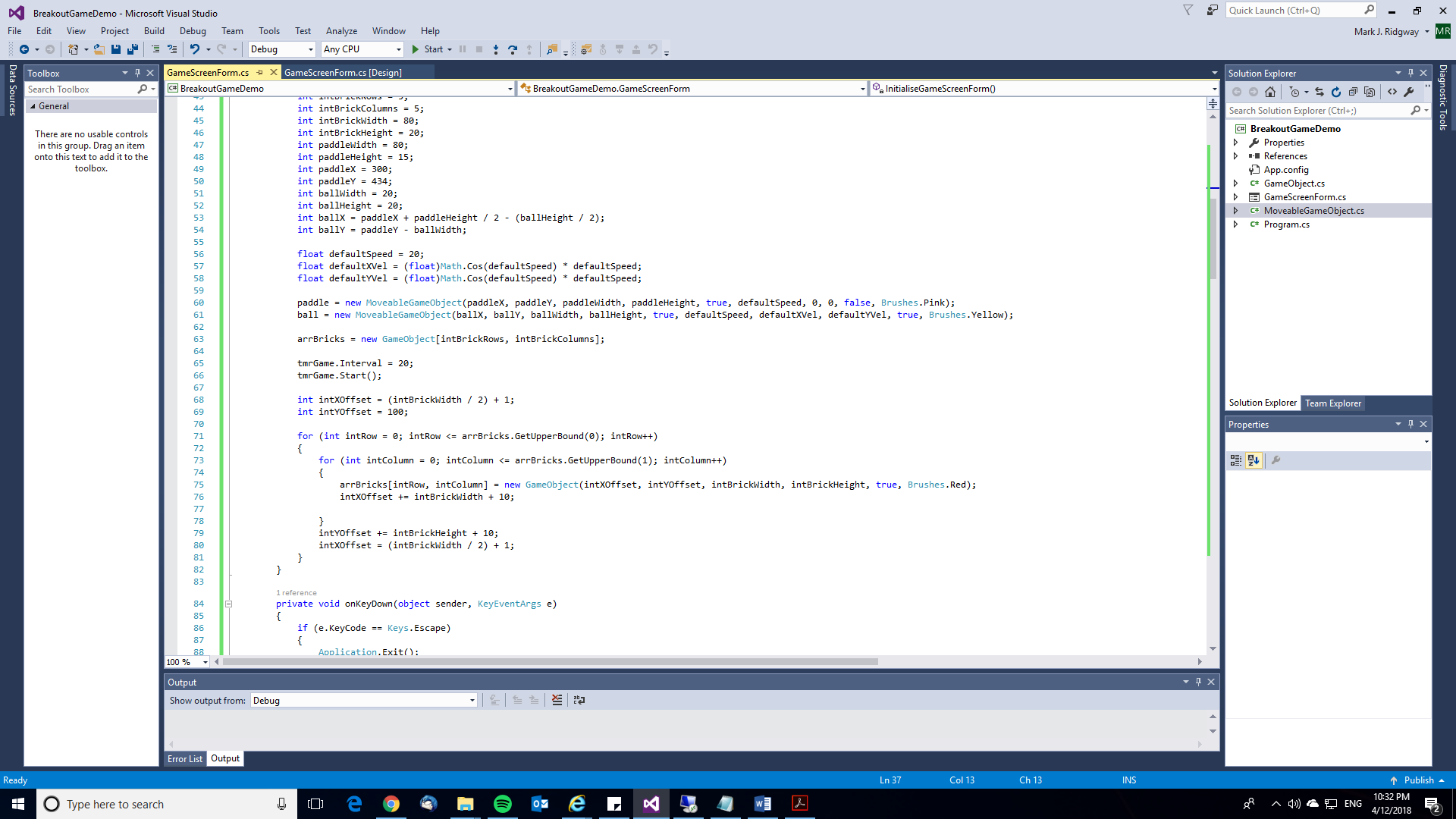
**Note: you would then repeat this evidence of iterative development until you have a working game. Then you would complete the Evaluation section of the coursework by carrying out post development system testing and usability testing which could involve further iterative development if errors are found. Finally you would write the last three parts of the evaluation: success of the solution, describe the final product, maintenance and development.**

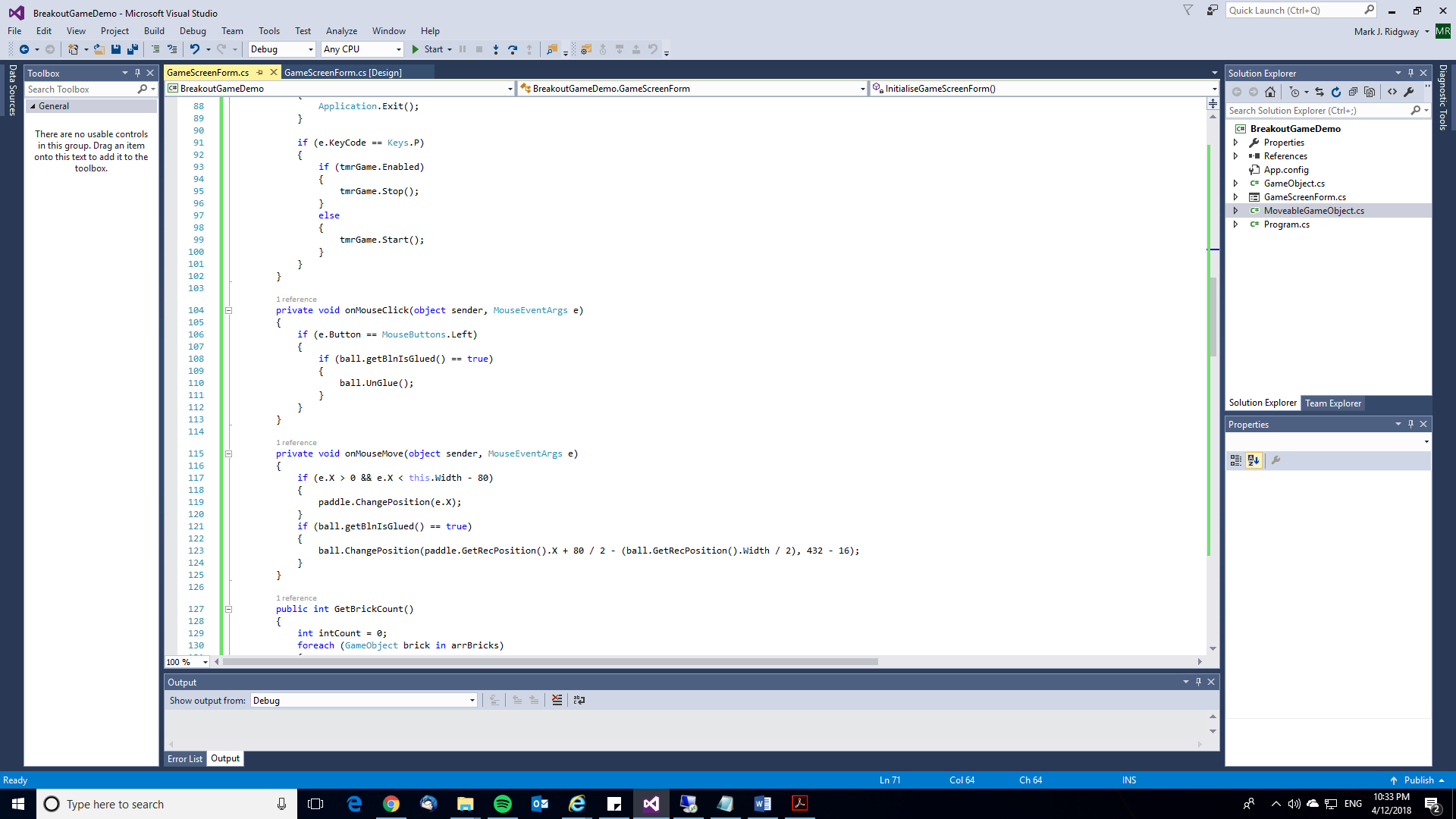
# Final code to get game working

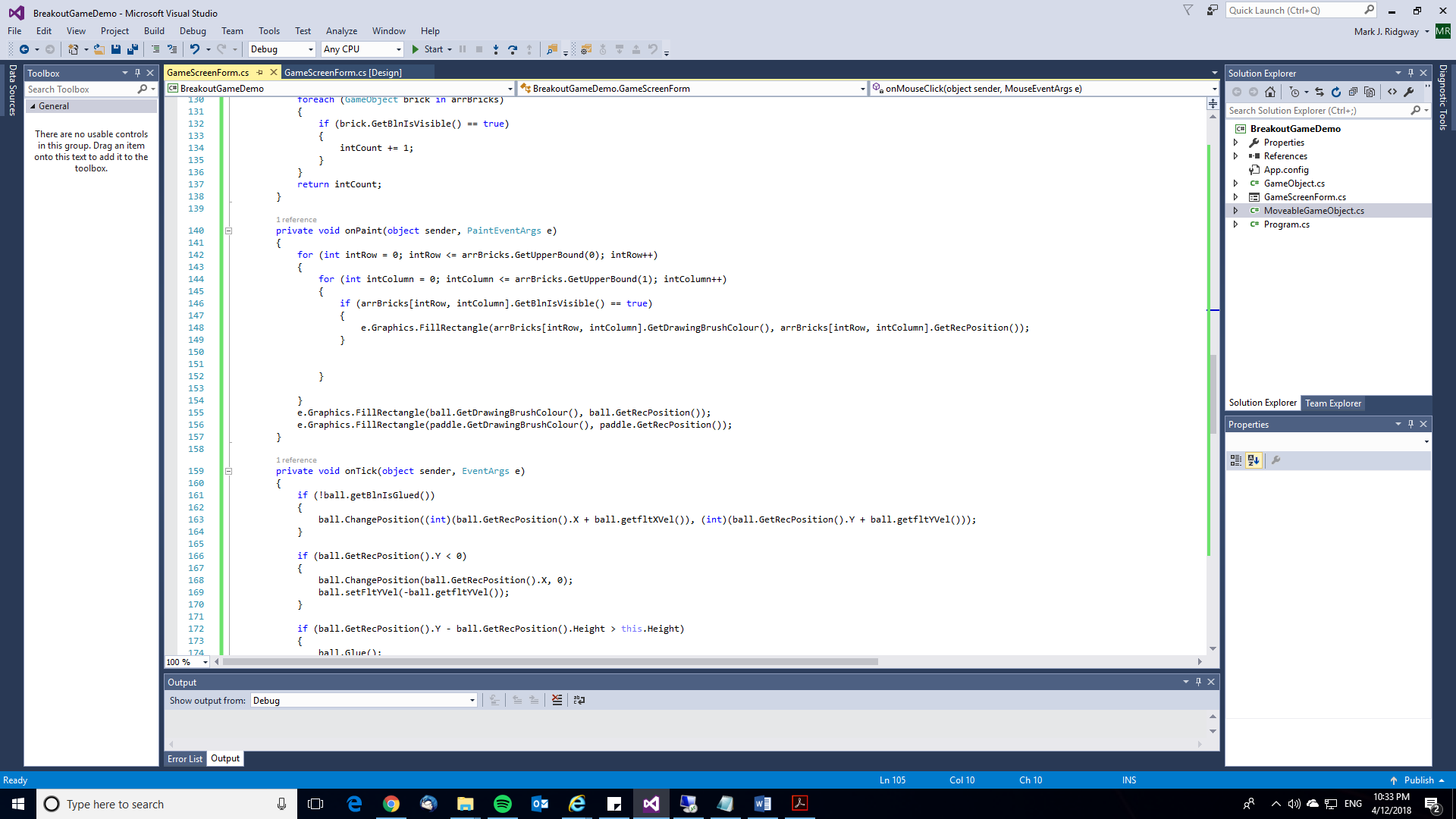
I’ve run out of time creating these materials so below is just the code for the Windows Form Class BreakOutGameScreen to get your game working. **Note: you would be expected to provide full evidence of your iterative development in the format I have shown in iteration 1 and 2.**

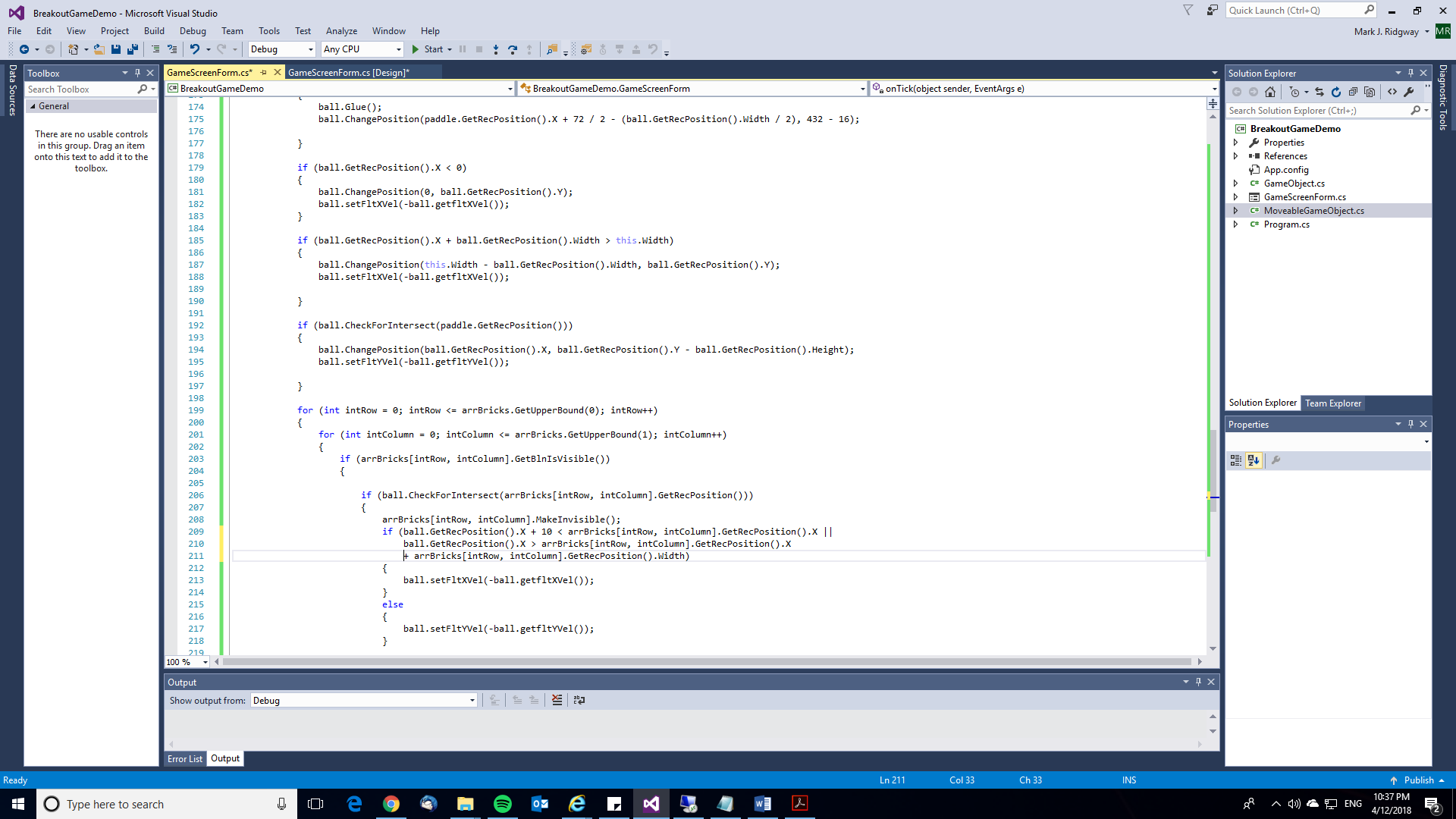
It would be good practice to study the code below and write detailed requirements for this iteration, draw a structure diagram breaking down the components of the solution, produce a usability testing plan, a class diagram for BreakOutGameScreen showing the class name, properties and methods, a during development testing plan, a post development system testing for function and robustness plan and carry out the actual during development testing filling in the extra column on the testing plan and showing proof of test results (a screenshot of the test explorer window as described.   
**Note: I didn’t say to write pseudocode from the actual code – you must write your pseudocode before you actually code. It doesn’t matter if you actual code ends up working differently to what you originally planned!**

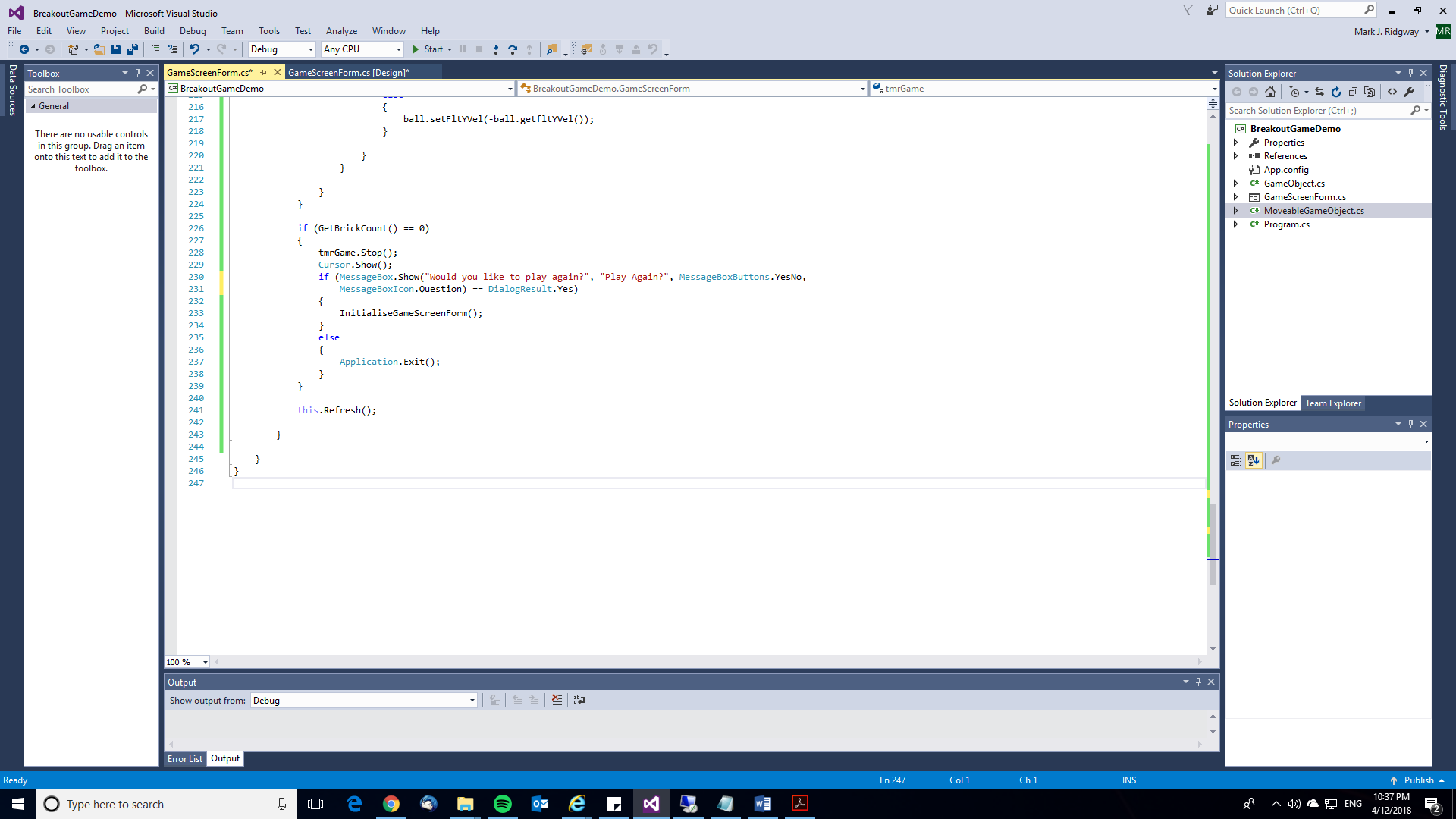












**This game is not complete. Spend some time developing the game further. It would be ideal to practice the iterative approach but as time is short don’t feel you have to:**

* Sound / music
* Scoring (display in a label on the screen)
* Game instructions
* Levels with different arrangements of bricks (or random arrangement)
* High score table (saved to disk e.g. variable length text file or direct access file)