ELEC362 Assessment 2

Ben Hague (201146260)

Department of Electrical Engineering and Electronics

17 December 2018

Abstract

This Report outlines the task given in ELEC362 Assignment 2 and discusses the solution to the problem and how the code has been constructed. We then move on to show the testing of the program and how it successfully meets the specification.

Declaration

I confirm that I have read and understood the University's definitions of plagiarism and collusion from the Code of Practice on Assessment. I confirm that I have neither committed plagiarism in the completion of this work nor have I colluded with any other party in the preparation and production of this work. The work presented here is my own and in my own words except where I have clearly indicated and acknowledged that I have quoted or used figures from published or unpublished sources (including the web). I understand the consequences of engaging in plagiarism and collusion as described in the Code of Practice on Assessment (Appendix L).

Contents

Introduction

Assignment 2 requires me to Design and Make an Arachnoid styled game. The Key obstacles in this task is to implement through the MFC Framework and to Correctly implement changeable settings for the ball speed grid size and paddle colour. The MFC Framework was introduced by Microsoft in 1992 as an object orientated method for creating software for windows. It is not suited for creating games and is much more appropriate for creating document editing programs or similar. In reflection I think that it would be more appropriate in the future if a specialised games framework able to handle drawing objects and shapes more efficiently was used rather than MFC.

Specifications

1. It should have multiple bricks types, each type has a colour and score assigned to it.
2. The game should have at least 3 rows of bricks, the type of these bricks should be assigned randomly.
3. It should have a score counter.
4. When the ball is lost, a message stating “Game Over: your score is “then the score should be displayed.
5. The control of the paddle should be either by the mouse or by the keyboard (both functionalities must be available).
6. The game should have the ability to save and load a game.
7. The game should have menus and dialogs allowing the user to set the speed of the ball, the grid size, and the colour of the paddle.
8. The game should have a “reset” button allowing the game to start from beginning and resting the score to 0.
9. The “about” dialog should have your full name and student ID number.

Methodology and Implementation

There are several simple methods used in this program relating individually to the different types of movement. I have separated out the 2 key methods needed for gameplay into the flowcharts in Figure 1 these overviews how the ball movement occurs and how the paddle movement occurs.

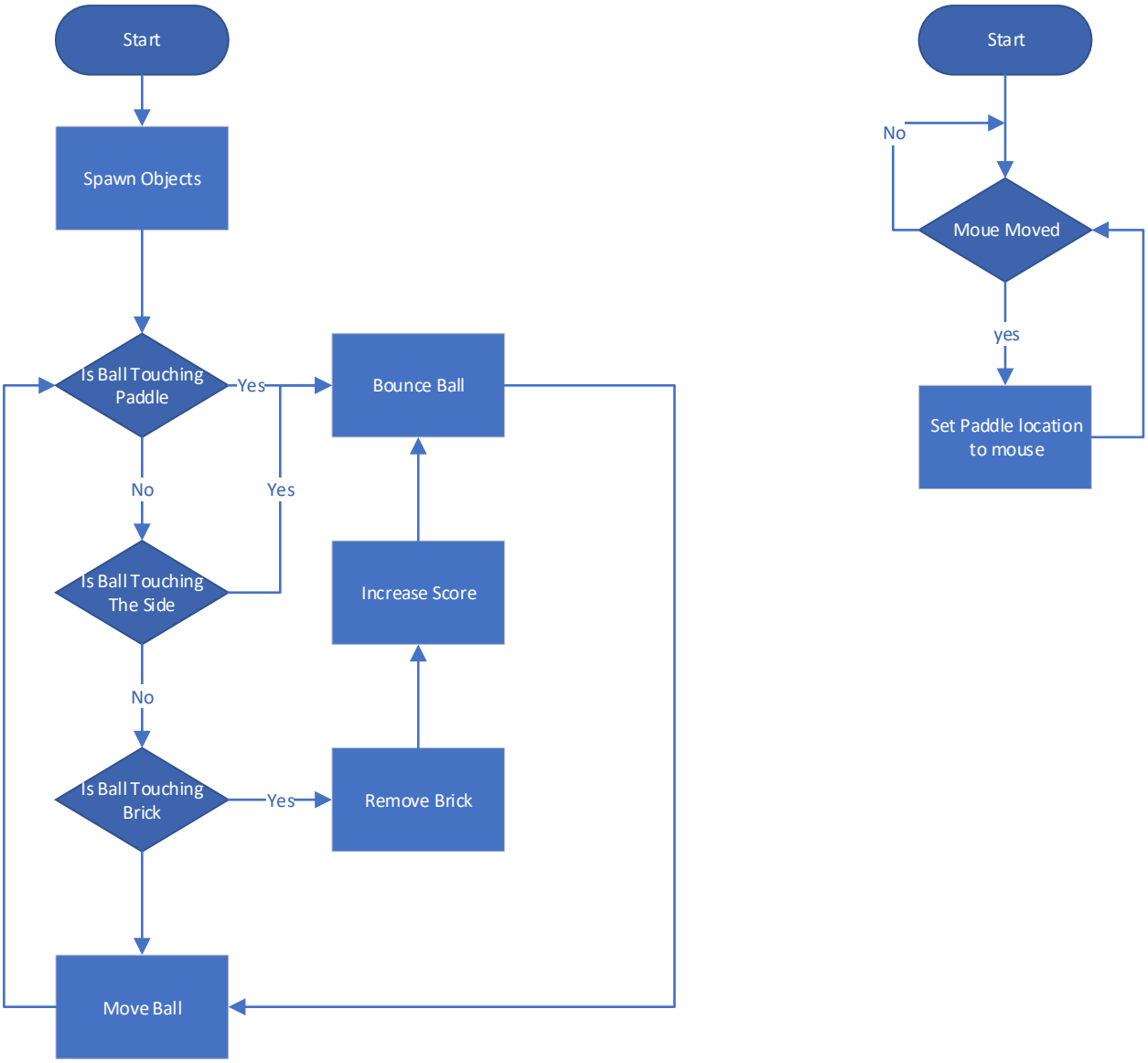

Figure 1: Flowchart for ball and paddle movements

There are a few key mathematical methods for this system

Bounce – to effectively bounce we need to invert the Velocity in the direction perpendicular to the surface we do this by multiplying this Value by -1.

Scoring – the scoring algorithm uses the Value of a brick cubed as the score added this means bricks can be worth between 1 and 125.

Other Key Mechanisms

Pausing – to pause the program we halt the movement of the Ball as this will prevent either the blocks being destroyed or the ball hitting the bottom of the play area causing game over.

This does however cause a bug when changing settings when paused where the program crashes.

Keyboard Paddle Movement - This is an always on feature of my code, this triggers an event which moves the paddle in the respective direction by 20 Pixels.

Objects and classes

In Figure 2 you can see a full Class Diagram for my Solution to the task. These details a full list of the classes and objects included and their methods and their variables

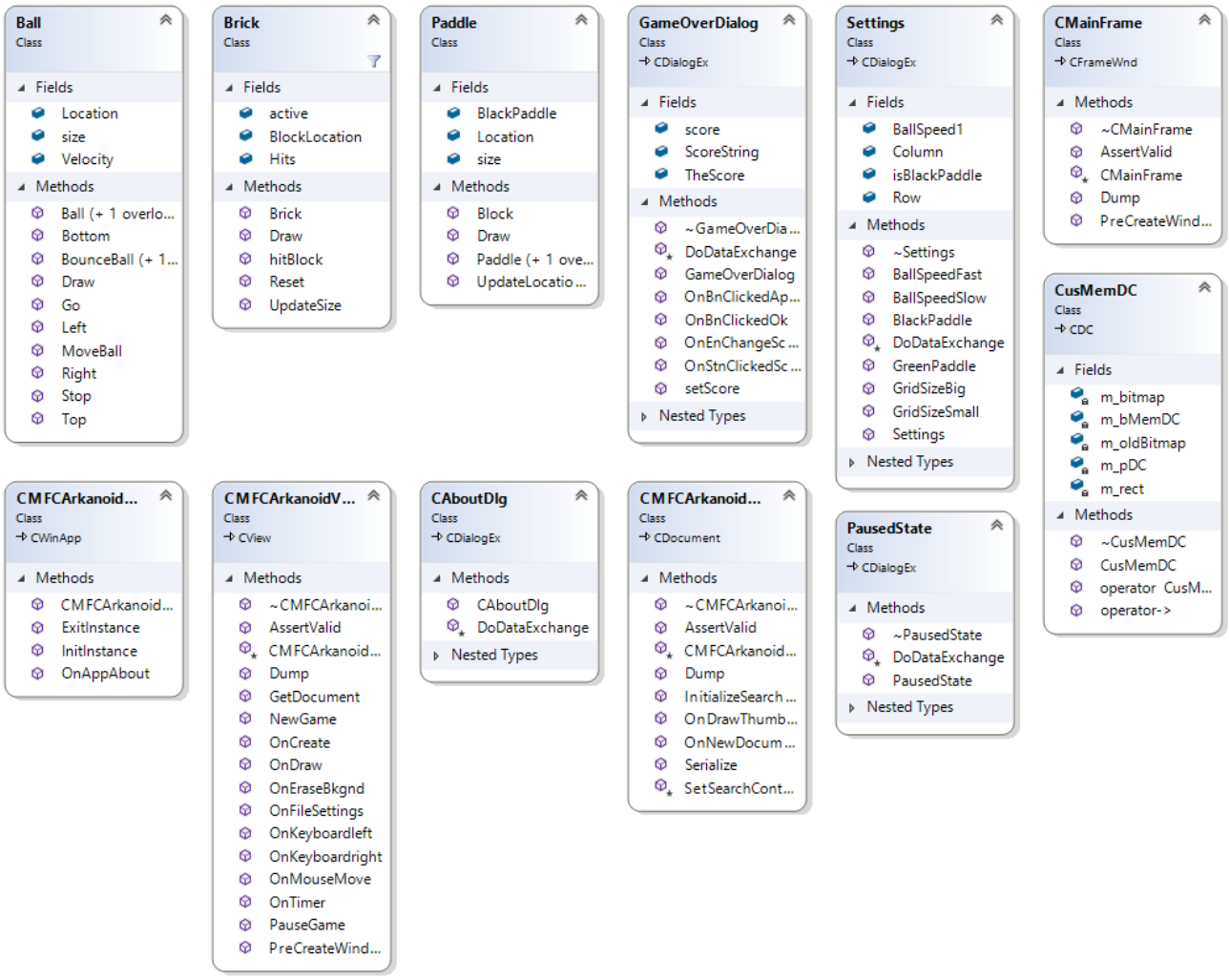

Figure 2: Class Diagram

In Table 1 you can see each of the class names along with a summary of what is contained within them and what they do along with if they are generated automatically by visual studio or if I created them.

Table 1: Classes and Descriptions

Creation	Class	Description
Manual	Ball	Declares the object type of Ball, this is used to make, draw and move the ball around the screen. The BounceBall Function handles Bouncing the ball
Manual	Brick	Declares the object type of Brick. This Draws the Bricks on the screen individually, calculates the location of the brick and can either be active or not. If it is not active than all interactions with the brick are ignored.
Manual	Paddle	Declares the Object of type Paddle. This handles the drawing of the paddle, paddle location, size and what colour it is drawn in.
Manual	GameOverDialog	The GameOverDialog handles the Dialog box which appears in a game over scenario. It is generated when the game is lost and given the score which is then displayed in the window.
Manual	Settings	The Settings Class handles the Settings window. This is generated when the settings button in the menu is clicked. This handles the radio button events and stores the result.
Automatic	CMainFrame	This acts as the main function of a simple program. The CMainFrame Class handles the order of execution and the passing of event messages
Automatic	CMFCArkanoidApp	This handles the About event, new file event and the Overall key program variables such as the application ID
Automatic	CMFCArkanoidView	This is an automatically generated Class which handles the Window open on the screen and makes it visible, handles the drawing on it and events between states
Automatic	CMFCArkanoidDoc	This Handles opening and closing files, as I have not impliemented this functionality this class remains unused
Automatic	CAboutDlg	This lives within the MFCArkanoid.cpp file, it handles the event where the about button is pressed and triggers the about app
Manual	PausedState	The PausedState class handles the Pause dialog, an object is created upon clicking the spacebar.
Manual-ish	CusMemDC	This is a class written by Keith Rule on Code Project (https://www.codeproject.com/Articles/33/%2FArticles%2F33%2FFlicker-Free-Drawing-In-MFC) this is used to combat the screen flicker

GUI design and role of Event Handlers

For the GUI design all necessary buttons are held within the menu and each of these triggers an event which is then handled to produce the output.

Table 2: Custom Event Handlers to Functions Map

Action	Event	Handler
New Game	ID_FILE_NEW	` CMFCArkanoidView::NewGame
Settings	ID_FILE_SETTINGS	CMFCArkanoidView::OnFileSettings
About MFCArkanoid	ID_APP_ABOUT	CMFCArkanoidApp::OnAppAbout
Space Bar	ID_KEYBOARDSPACE	CMFCArkanoidView::PauseGame
Right Arrow Key	ID_KEYBOARDRIGHT	CMFCArkanoidView::OnKeyboardright
Left Arrow Key	ID_KEYBOARDLEFT	CMFCArkanoidView::OnKeyboardleft
Ball Speed Slow Radio Button	IDC_RADIO1	Settings::BallSpeedSlow
Ball Speed Fast Radio Button	IDC_RADIO2	Settings::BallSpeedFast
Grid Size Small Radio Button	IDC_RADIO3	Settings::GridSizeSmall
Grid Size Big Radio Button	IDC_RADIO4	Settings::GridSizeBig
Black Paddle Radio Button	IDC_RADIO5	Settings::BlackPaddle
Green Paddle Radio Button	IDC_RADIO6	Settings::GreenPaddle

Each of these Works as required to allow the game to correctly execute.

Table 3: Explanation of Figures, contains a list of figures shown below and explans what they are showing and which specification this meets.

Table 3: Explanation of Figures

Figure	Explanation	Specificatrion point
Figure 3: Demonstration of basic Functionality	This shows the Game when it first starts, this is the standard gameplay screen. As you can see it shows 3 rows of blocks with 10 Columns with 5 different brick types, each with a different score. The Score Counter Features Under the paddle at the bottom of the screen	1,2,3
Figure 4: Demonstration of Big Grid Mode	Big Grid mode increases the number of Columns to 20 and Rows to 5, this demonstrates big grid mode and the output from that	7
Figure 5: Demonstration of Paused Game	This shows the paused game dialog which offers you to continue playing or access the settings. This is accessed by clicking the space key.	Surplus to
Figure 6: Demonstration of settings	This shows the possible settings and how they can be changed. It shows how you can change Ball Speed, paddle Colour and grid size	7
Figure 7: Demonstration of settings applied in Figure 6	The settings that are clicked in Figure 6 have now been applied to the game and this is the output	7
Figure 8: Demonstration of Game Over Screen	This is the screen showing the score, that indicates that you have lost at the game	4
Figure 9: Demonstration of New Game (Reset) Button	The New Game Button Resets the game sets the score to 0 and re-draws the Blocks	8
Figure 10: Demonstration of About Screen	The About Screen, Shows Name, Student ID number and a Cheeky selfie for good measure.	9

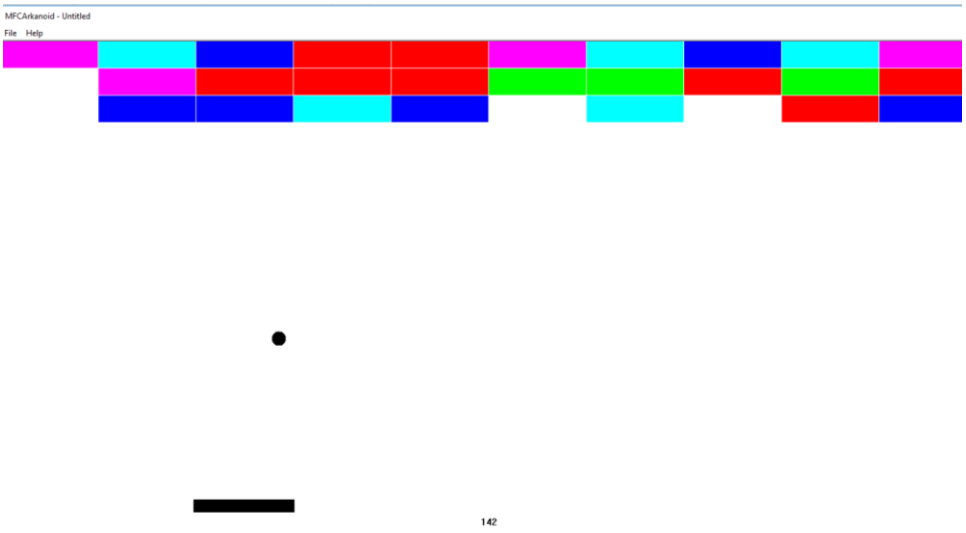

Figure 3: Demonstration of basic Functionality

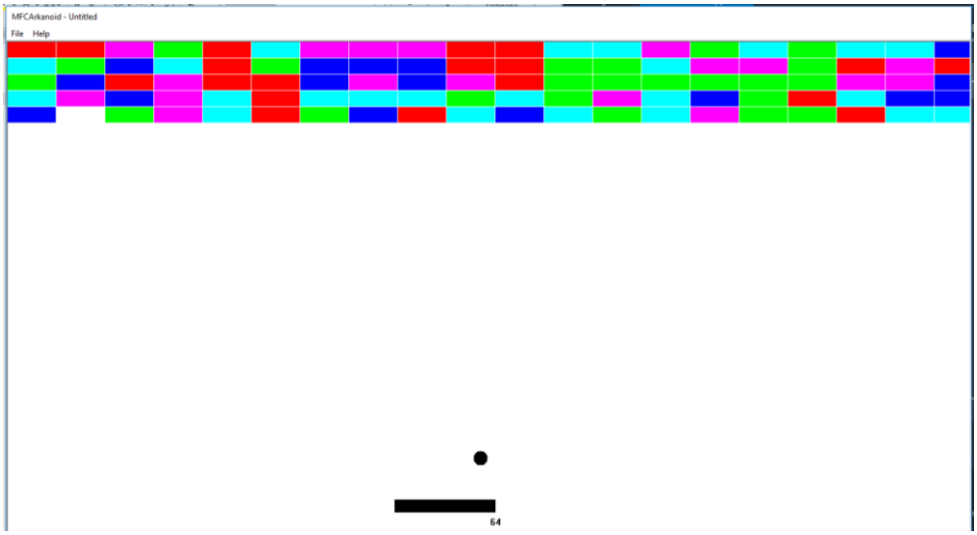

Figure 4: Demonstration of Big Grid Mode

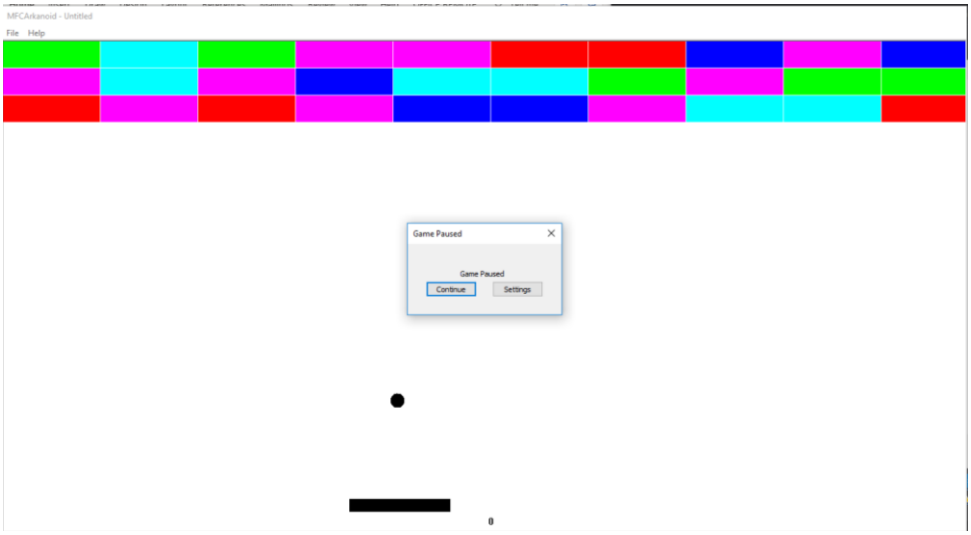

Figure 5: Demonstration of Paused Game

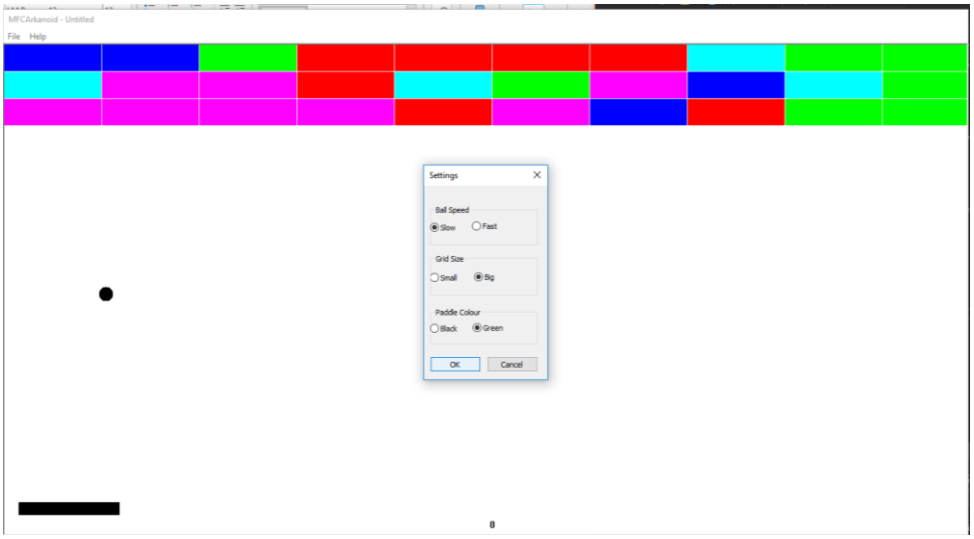

Figure 6: Demonstration of settings

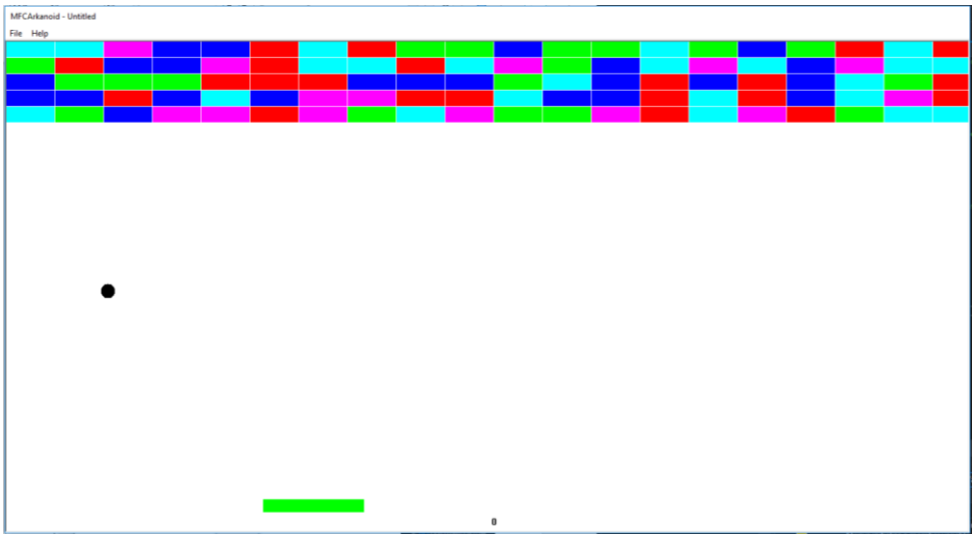

Figure 7: Demonstration of settings applied in Figure 6

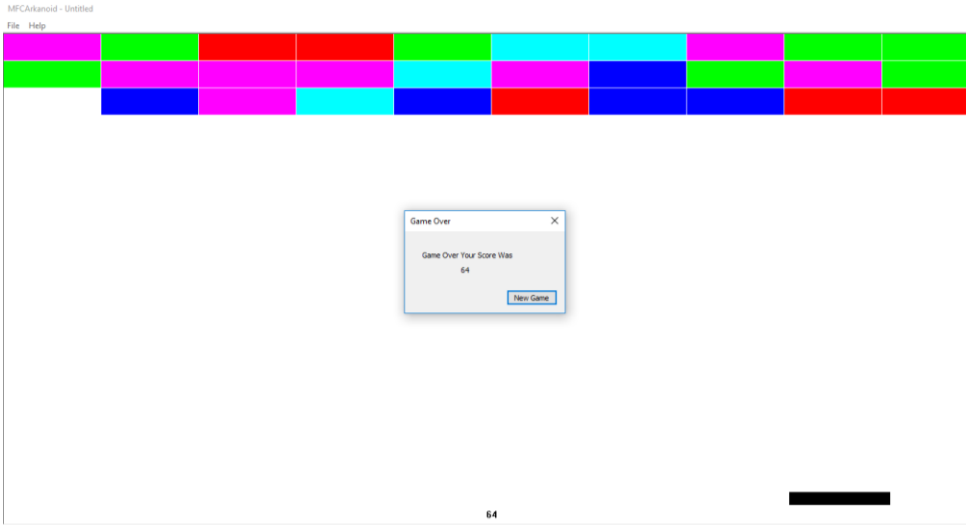

Figure 8: Demonstration of Game Over Screen

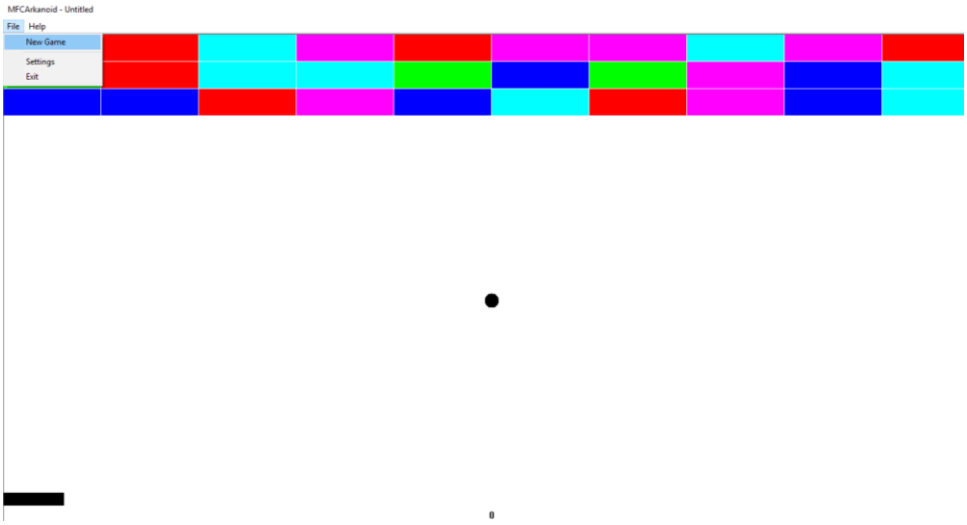

Figure 9: Demonstration of New Game (Reset) Button

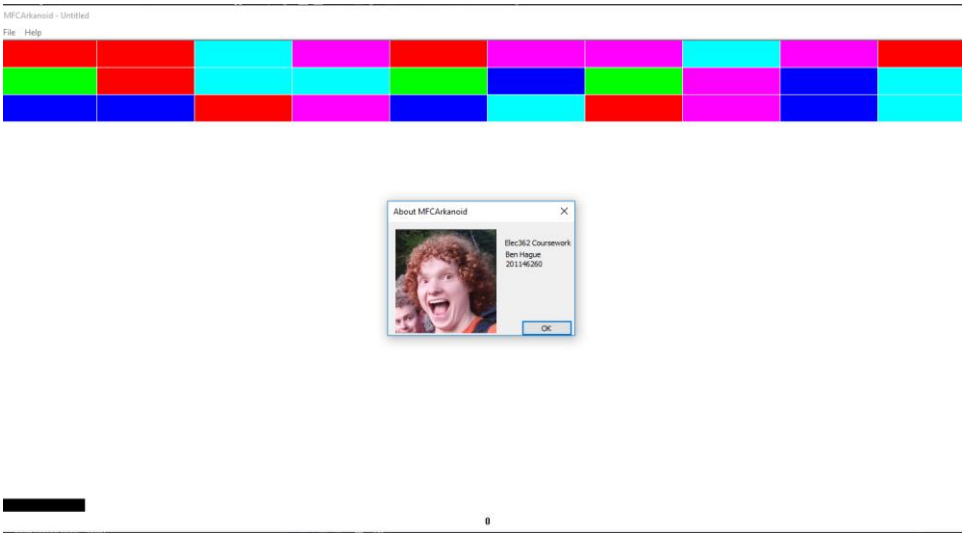

Figure 10: Demonstration of About Screen

Source code

Due to the Sheer Number of classes and content of the code I have included a reference as to what is contained in each header files and script files within the zip file submitted. The code Included in appendix 1 is the most Code, Ball.cpp, Paddle.cpp, Brick.cpp, CMFCArkanoidView.cpp are all included and are the most likely to be of interest. Each of these files has its respective header file where all the functions are declared. The other files are all contained within the uploaded ZIP file.

Summary and Specification Review

To test the program, I have run a range of possible scenarios. There is one clear bug I am aware of at the current moment where if you change some settings whilst the game is paused then the application crashes. Other than this the program works effectively using minimal Ram and CPU usage as expected in the section above I have outlined the functionality of the program and how it runs. Below I outline how it meets the required specifications.

1. It should have multiple bricks types, each type has a colour and score assigned to it.

   The game has 5 brick types Each one is has a different hit value and colour, when each brick is hit it demotes it a brick type, when the brick type is 0 the brick disappears.
2. The game should have at least 3 rows of bricks, the type of these bricks should be assigned randomly

   The Game has a starting setting with 10*3 bricks, this can be increased using the settings dialog
3. It should have a score counter.

   This sits underneath the paddle in the game
4. When the ball is lost, a message stating “Game Over: your score is “then the score should be displayed.

   When you lose the game the message states Game Over and says the score it also has a new game function button
5. The control of the paddle should be either by the mouse or by the keyboard (both functionalities must be available).

In normal control Mode the Game uses the X Value of the point to determine where the paddle goes, For the keyboard we trigger an event which moves the paddle by a defined distance when the specified keys are pressed

6. *The game should have the ability to save and load a game.

   Due to the volume of work needed to be done in this time this requirement has been missed.

7. The game should have menus and dialogs allowing the user to set the speed of the ball, the grid size, and the colour of the paddle.

   The Game has a settings Dialog where the user can set the Ball Speed, Grid Size and Paddle Colour

8. The game should have a “reset” button allowing the game to start from beginning and resting the score to 0.

   By Clicking File New Game is resets the game with new blocks and resets the ball to the 0 position.

9. The “about” dialog should have your full name and student ID number.

   The About dialog contains my full name, the module code, and my student id number along with a selfie.

Conclusion

The biggest challenge with this assignment was understanding how to use the MFC Framework within 3 weeks whilst balancing other university work and tests. Other challenges were particularly related to interpreting the input from the dialog window. I feel if this was to be redone I would look for a more appropriate framework to build my application on.

Appendix

Code from Ball.cpp

```
#include "stdafx.h"

#include "Ball.h"
	Ball::Ball() {  //Constructor Default defines a ) velocity , Location and 0 size, this acts as a placeholder for when the ball has been given its settings later
		size = 0;
		Velocity[0] = 0;
		Velocity[1] = 0;
		Location = CPoint(0,0);
	}
	Ball::Ball( int Xdir, int Ydir, CPoint StartLocation) { //define the start variables from the constructor

		size = 20;
		Velocity[0] = Xdir;
		Velocity[1] = Ydir;
		Location = StartLocation;
	}
	void Ball::MoveBall() // move the x and y of the ball by the velocity

	{
		Location.x += Velocity[0];
		Location.y += Velocity[1];
	};
	void Ball::BounceBall(bool Horozontal) //Bounceball based on the angle of the surface it hits, Horosontal or Vertical
	{
		if (Horozontal)
		{
			Velocity[0] *= -1; //bounce in the vertical space by inverting Y component of movement
			Velocity[1] *= 1;
		}
		else
		{
			Velocity[1] *= -1; //Bounce in Horosontal space by inverting the x component of movement
			Velocity[0] *= 1;

		}
	};
	void Ball::Draw(CDC* pDC) // Input the pem object, produce a drawn output on the screem by drawing the paddle
	{
		CPen BallPen;
		BallPen.CreatePen(PS_SOLID, 2, RGB(0, 0, 0));
		CBrush BallBrush{ RGB(0, 0, 0) };
		CPen* pOldPen{ pDC->SelectObject(&BallPen) };
		CBrush* pOldBrush{ pDC->SelectObject(&BallBrush) }; // Make pen and brush in black then select them and draw the ball at the location
		pDC->Ellipse((Location.x - size / 2), (Location.y - size / 2), (Location.x + size / 2), (Location.y + size / 2));
	};
	CPoint Ball::Bottom() //Returns the Bottom of the ball as a cpoint
	{
		return CPoint((Location.x), Location.y + size / 2);
```

```
}
CPoint Ball::Top()//Returns the Top of the ball as a cpoint
{
	return CPoint((Location.x), Location.y - size / 2);
}
CPoint Ball::Left()//Returns the Left of the ball as a cpoint
{
	return CPoint((Location.x - size / 2), Location.y );
}
CPoint Ball::Right()//Returns the Right of the ball as a cpoint
{
	return CPoint((Location.x + size / 2), Location.y);
}

void Ball::Stop(CPoint StopLocal) //Stops the ball at the set location
{
	Location = StopLocal;
	Velocity[0] = 0;
	Velocity[1] = 0;
}
void Ball::Go(int Ballspeed) //Starts the ball at the specified speed
{
	Velocity[0] = -Ballspeed;
	Velocity[1] = Ballspeed;
}
```

Code from Paddle.cpp

```
#include "stdafx.h"

#include "Paddle.h"
	Paddle::Paddle(int Size, CPoint StartLocation) //Constructor sets the location, and
size
	{
		Location = StartLocation;
		size = Size;
	};
	Paddle::Paddle() { //Default Paddle Size Declared
		size = 150;
	}
	CRect Paddle::Block() // Return the rectangle which is the paddle
	{
		return CRect(Location.x - size / 2, Location.y - 10, Location.x + size / 2,
Location.y + 10);
	}
	void Paddle::Draw(CDC* pDC) // draw the paddle
	{
		CPen PaddlePen; //create the pen object
		PaddlePen.CreatePen(PS_NULL, 2, RGB(0, 0, 0)); //Set Null Pen Colour
		CPen* pOldPen{ pDC->SelectObject(&PaddlePen) }; //Select the paddle Pen
		CBrush BlackBrush{ RGB(0, 0, 0) }; // Create the black Brush
		CBrush PaddleBrush{ RGB(0, 255, 0) };//create the green brush
		CBrush* pOldBrush{ pDC->SelectObject(&PaddleBrush) }; //Select te green brush
		if (BlackPaddle) //if black paddle
		{

			CBrush* pOldBrush{ pDC->SelectObject(&BlackBrush) }; // select the black
brush
		}
		pDC->Rectangle(Block()); //draw block
	}

	void Paddle::UpdateLocation(CPoint Place)
	{
		Location.x = Place.x; // Set the x location to the specified location given.
	}
	void Paddle::UpdateLocation(int Distance)
	{
		Location.x += Distance; // Set the x location to the specified location given.
	}
```

Code from Brick.cpp

```
#pragma once
#include "stdafx.h"
#include <time.h>
#include "Brick.h"

	Brick::Brick(int BlockW, int BlockH, int X, int Y) // Brick constuctor sets the bnrick area (BrickLocation) and the number of hits taken to destroy the brick
	{
		BlockLocation = CRect(X*BlockW, Y*BlockH, (X + 1)*BlockW, (Y + 1)*BlockH);

		Hits = (rand() % 5)+1;
	};
	void Brick::UpdateSize(int BlockW, int BlockH, int X, int Y) // Brick constuctor sets the bnrick area (BrickLocation) and the number of hits taken to destroy the brick
	{
		BlockLocation = CRect(X*BlockW, Y*BlockH, (X + 1)*BlockW, (Y + 1)*BlockH);
	};
	void Brick::Reset() //Reset, ReRandomises the brick and resets the active Flag for the brick, The if a brick is not active it cannot be collided with
	{
		active = true;
		Hits = (rand() % 5) + 1;
	}

	void Brick::hitBlock() // when the block is hit remove one from the hits. then if the hits <= 0 make the active Flag False
	{
		Hits = 0;
		if (Hits <= 0)
		{
			active = false;
		}
	}
	void Brick::Draw(CDC*pDC) // Draw uses the Hits and draws a brick of set colour depending on the number of hits remaining.
	{
		int colour[3] = { 255,255,255 };
		//Set the colour for each value of brick
		switch (Hits) {
		case 1: //Blue
			colour[0] = 0;
			colour[1] = 0;
			colour[2] = 255;
			break;
		case 2: //Green
			colour[0] = 0;
			colour[1] = 255;
			colour[2] = 0;
			break;
		case 3://Red
			colour[0] = 255;
			colour[1] = 0;
			colour[2] = 0;
			break;
		case 4: //Pink
			colour[1] = 0;
			colour[0] = 255;
			colour[2] = 255;
			break;
		case 5: // cyan
```

```
                colour[0] = 0;
                colour[1] = 255;
                colour[2] = 255;
                break;
            }

            CPen BrickPen; //Create the pen
            BrickPen.CreatePen(PS_NULL, 2, RGB(0, 0, 0));//set to invisible
            CBrush BrickBrush{ RGB(colour[0], colour[1], colour[2]) }; // set the coloured
brush
            CPen* pOldPen{ pDC->SelectObject(&BrickPen) }; //select the pen
            CBrush* pOldBrush{ pDC->SelectObject(&BrickBrush) }; //select the brush
            pDC->Rectangle(BlockLocation);//Draw the Block
        };
```

Code from MFCArkinoidView.cpp

```
// MFCArkanoidView.cpp : implementation of the CMFCArkanoidView class
//

#include "stdafx.h"
// SHARED_HANDLERS can be defined in an ATL project implementing preview, thumbnail
// and search filter handlers and allows sharing of document code with that project.
#ifndef SHARED_HANDLERS
#include "MFCArkanoid.h"
#endif

#include "MFCArkanoidDoc.h"
#include "MFCArkanoidView.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#endif

// Include the relevant header files
#include <vector>
#include "Ball.h"
#include "Paddle.h"
#include "Brick.h"
#include<time.h>
#include "memdc.h"
#include "GameOverDialoig.h"
#include "Settings.h"
#include "PausedState.h"

//Initilise the objects for use throughout the methods in this class
Ball TheBall; // Make a Ball Object
Paddle ThePaddle; // Make a Paddle Object
std::vector <std::vector <Brick>> Blocks; //Make an vector of a vector of blocks

// Determine the key gameplay variables
int Rows = 3;
int Columns = 10;
int BallSpeed = 2;

// Create an object to represent the screen
CRect Screen;

// and a Variable to for the score
int Score;
// CMFCArkanoidView

IMPLEMENT_DYNCREATE(CMFCArkanoidView, CView)

// Declare the message handlers
BEGIN_MESSAGE_MAP(CMFCArkanoidView, CView)
	ON_WM_TIMER()
	ON_WM_ERASEBKGND()
	ON_WM_MOUSEMOVE()
	ON_WM_CREATE()
	ON_COMMAND(ID_KEYBOARDLEFT, &CMFCArkanoidView::OnKeyboardleft)
	ON_COMMAND(ID_KEYBOARDRIGHT, &CMFCArkanoidView::OnKeyboardright)
	ON_COMMAND(ID_FILE_SETTINGS, &CMFCArkanoidView::OnFileSettings)
	ON_COMMAND(ID_FILE_NEW, &CMFCArkanoidView::NewGame)
	ON_COMMAND(ID_KEYBOARDSPACE, &CMFCArkanoidView::PauseGame)
END_MESSAGE_MAP()
```

```
// CMFCArkanoidView construction/destruction

CMFCArkanoidView::CMFCArkanoidView() noexcept
{
	// TODO: add construction code here

}

CMFCArkanoidView::~CMFCArkanoidView()
{
}

// Code For Pre Creation of the window, this is empty as all the code is executed once the
window is drawn to account for the window size
BOOL CMFCArkanoidView::PreCreateWindow(CREATESTRUCT& cs)
{
	// TODO: Modify the Window class or styles here by modifying
	//  the CREATESTRUCT cs

	return CView::PreCreateWindow(cs);
}

// CMFCArkanoidView drawing
// this is the Drawing Method, It contains all the code calculating the drawing, it is
reexecuted whenever the timer times out.
void CMFCArkanoidView::OnDraw(CDC* dc)
{
	CusMemDC pDC(dc); // this is a custom frame buffer, removes the flickering

	// This sets up the document in the Required format
	CMFCArkanoidDoc* pDoc = GetDocument();
	ASSERT_VALID(pDoc);
	if (!pDoc)
		return;
	//This Assigns the current Screen size in this iteration to the rectangle Screen
	CWnd* Res = this->GetParent();
	CRect Screen;
	Res->GetWindowRect(&Screen);

	//start the timer to refresh the screen every 5ms
	SetTimer(1, 5, NULL);

	//define the BlockWidth and Bock Height in this drawing
	int BlockWidth = (Screen.Width()) / Columns;
	int BlockHeight = 60 / Rows;

	//Draw the Ball in its current position and then move the ball. This means we are
calculating the next visable change in the objects
	TheBall.Draw(pDC);
	TheBall.MoveBall();

	//Declare the Game area, this is the area of the screen where the game takes place, we
remove the bottom 100pixels for the score
	CRect GameArea(0, 0, Screen.Width(), Screen.Height()-100);

	//Check if the base of the ball exits the game area, if it does then show the game
over Prompt
	if (!PtInRect(GameArea, TheBall.Bottom()))
	{
		GameOverDialog GameOver;
		GameOver.setScore(Score);
```

```
			TheBall.Stop(CPoint(CPoint(Screen.Width() / 2, Screen.Height() / 2)));
			if (GameOver.DoModal() == IDOK) {
				NewGame();
			};
		}
		//Check if the Ball is touching the sides of the Game Area if so, Bounce
		if (!PtInRect(GameArea, TheBall.Top()))
		{
			TheBall.BounceBall(false);
		}
		if (!PtInRect(GameArea, TheBall.Right()) || !PtInRect(GameArea, TheBall.Left()))
		{
			TheBall.BounceBall(true);
		}
		// For each of the blocks that is spawned, Update the size and location of the block
		//then check if the Ball has collided, if so then bounce the ball in its respective
direction
		for (int x = 0; x < Columns; x++) {
			for (int y = 0; y < Rows; y++) {
				Blocks[x][y].UpdateSize(Screen.Width() / Columns, (120 / Rows), x, y);
				if (Blocks[x][y].active) {
					if (PtInRect(Blocks[x][y].BlockLocation, TheBall.Top()) ||
PtInRect(Blocks[x][y].BlockLocation, TheBall.Bottom()))
					{
						Score += pow(Blocks[x][y].Hits, 3);
						Blocks[x][y].hitBlock();
						TheBall.BounceBall(false);
					}
					if (PtInRect(Blocks[x][y].BlockLocation, TheBall.Left()) ||
PtInRect(Blocks[x][y].BlockLocation, TheBall.Right()))
					{
						Score += pow(Blocks[x][y].Hits, 3);
						Blocks[x][y].hitBlock();
						TheBall.BounceBall(true);
					}
					Blocks[x][y].Draw(pDC);
				}

			};
		};
		// Draw the paddle and check for a collision, if its collided, bounce
		ThePaddle.Draw(pDC);
		if (PtInRect(ThePaddle.Block(), TheBall.Bottom()))
		{
			TheBall.BounceBall(false);
		};
		// Set the area for the score and print the score in that rectangle
		CRect ScoreRect(0, Screen.Height() -85, Screen.Width(), Screen.Height());
		CString ScoreText;
		ScoreText.Format(_T("%i"), Score);
		pDC->DrawText(ScoreText, ScoreRect, DT_CENTER);

}


// CMFCArkanoidView diagnostics

#ifdef _DEBUG
void CMFCArkanoidView::AssertValid() const
{
		CView::AssertValid();
}
```

```
void CMFCArkanoidView::Dump(CDumpContext& dc) const
{
	CView::Dump(dc);
}

CMFCArkanoidDoc* CMFCArkanoidView::GetDocument() const // non-debug version is inline
{
	ASSERT(m_pDocument->IsKindOf(RUNTIME_CLASS(CMFCArkanoidDoc)));
	return (CMFCArkanoidDoc*)m_pDocument;
}
#endif //_DEBUG

void CMFCArkanoidView::NewGame() {

	//Make The Bricks in a temporary file and overwrite the blocks
	std::vector<std::vector<Brick>> tmpBlocks;
	for (int x = 0; x < Columns; x++) {
		std::vector<Brick> Row;
		for (int y = 0; y < Rows; y++) {
			Brick tmp((Screen.Width() / Columns), (120 / Rows), x, y);
			Row.push_back(tmp);
		};
		tmpBlocks.push_back(Row);
	};
	Blocks = tmpBlocks;
	//set the score to 0
	Score = 0;
	//Set the Location to the Center Point and Launch the Ball
	TheBall.Location = CPoint(Screen.Width() / 2, Screen.Height() / 2);
	TheBall.Go(BallSpeed);
}

// CMFCArkanoidView message handlers


void CMFCArkanoidView::OnTimer(UINT_PTR nIDEvent)
{
	// When the timer expires Refresh the screen
	UpdateData(FALSE);

	Invalidate();
	CView::OnTimer(nIDEvent);
}


BOOL CMFCArkanoidView::OnEraseBkgnd(CDC* pDC)
{
	//Dont Refresh the background

	return false;
}


void CMFCArkanoidView::OnMouseMove(UINT nFlags, CPoint point) // Event Handler for mouse
Movement
{
// Get the point where the cursor is and store it as P
	CPoint p;
	GetCursorPos(&p);
	ScreenToClient(&p);
```

```
	//Update the X position of the paddle to the same as the mouse
	ThePaddle.UpdateLocation(p);
	CView::OnMouseMove(nFlags, point);
}

//
int CMFCArkanoidView::OnCreate(LPCREATESTRUCT lpCreateStruct) // Event Handler for post
window creation
{
	if (CView::OnCreate(lpCreateStruct) == -1)
		return -1;
	//Store the window Size as Screen
	CWnd* Res = this->GetParent();
	Res->GetWindowRect(&Screen);

	//Start Random Number Seed
	srand((unsigned)time(NULL));

	//Create the Ball and Paddle Objects
	TheBall = Ball(BallSpeed, -BallSpeed, CPoint(Screen.Width() / 2, Screen.Height() /
2));
	ThePaddle = Paddle(150, CPoint(Screen.Width()/2, Screen.Height()-100));
	//Use the New Game Method to start a new game
	NewGame();

	return 0;
}


void CMFCArkanoidView::OnKeyboardleft() // Event Handler for left keyboard button
{
	// When Pressed move the paddle 20 pixels left
	ThePaddle.UpdateLocation(-20);
}


void CMFCArkanoidView::OnKeyboardright() // Event Handler for Right Keyboard Button
{
	// When Pressed move the paddle 20 pixels right
	ThePaddle.UpdateLocation(20);
}


void CMFCArkanoidView::OnFileSettings() // Event Handler for settings button
{
	//Create Settings Screen Item
	Settings SetScreen;
	// Set the Variables to match the current System Variables
	SetScreen.BallSpeed1 = BallSpeed;
	SetScreen.Column = Columns;
	SetScreen.Row = Rows;
	SetScreen.isBlackPaddle = ThePaddle.BlackPaddle;
	// Make the Screen Appear, On exit Update the Variables
	if (SetScreen.DoModal() == IDOK)
	{
		Rows = SetScreen.Row;
		Columns = SetScreen.Column;
		BallSpeed = SetScreen.BallSpeed1;
		TheBall.Velocity[0] /= abs(TheBall.Velocity[0]);
		TheBall.Velocity[1] /= abs(TheBall.Velocity[1]);
		TheBall.Velocity[0] *= BallSpeed;
		TheBall.Velocity[1] *= BallSpeed;
```

```
			ThePaddle.BlackPaddle = SetScreen.isBlackPaddle;

		}
}


void CMFCArkanoidView::PauseGame() //Event Handler for SpaceBar
{
		PausedState Pause; //Create Paused Dialog
		//Stop the ball where it is
		TheBall.Stop(TheBall.Location);
		if (Pause.DoModal() == IDOK)
		{
			TheBall.Go(BallSpeed); // when Continue is clicked resume the balls motion.
		}

}
```

Task Sheet

Deadline: 14-Dec, Total mark: 50% of the module's mark

Arkanoid videogame:

This assignment is inspired by an old famous game knowns as "Arkanoid". This game was popular in the 80s and early 90s. In this game there are multiple lines of bricks at the upper third of the screen, and the aim of the game is to break these bricks using a ball that bounces back when it hits a brick or hits the left/right end of the window. The player moves a paddle at the bottom of the screen left and right to prevent the ball from touching the bottom end of the screen. As soon as the ball hits the bottom end of the screen, the game is over. A screen view of the classic game is shown in figure 1.

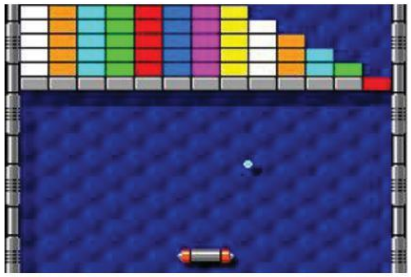

Figure 1: A screen view of Arkanoid.

In this assignment you are required to create a version of "Arknoid" game using MFC. The game should have the following specifications:

1. It should have multiple bricks types, each type has a colour and score assigned to it.
2. The game should have at least 3 rows of bricks, the type of these bricks should be assigned randomly.
3. It should have a score counter.
4. When the ball is lost, a message stating "Game Over: your score is "then the score should be displayed.
5. The control of the paddle should be either by the mouse or by the keyboard (both functionalities must be available).
6. The game should have the ability to save and load a game.
7. The game should have menus and dialogs allowing the user to set the speed of the ball, the grid size, and the colour of the paddle.
8. The game should have a "reset" button allowing the game to start from beginning and resting the score to 0.
9. The "about" dialog should have your full name and student ID number.

Deliverables of the project: All Deliverables should be uploaded to vital.

- Report.
- Executable program (must run in L3 Computer labs!).
- Source code (must compile on Visual Studio in the L3 Computer labs – please 'zip' the solution into one file).

Report specifications:

Section 1: Introduction

An introduction to the objectives of this assignment and an analysis of the project specification with a focus on the functionality of the program. The original problem specification should be included in the report as an appendix.

Section 2: Methodology and Implementation

Detailed description of any mathematics or physical mechanisms that are implemented in the event handlers and how they are implemented. Your explanation should be clear and concise. All variables or symbols must be explained when they appear in the report for the first time.

Section 3: Objects and classes

Describe what object and classes you are using and how they relate to each other.

Section 4: GUI design and role of Event Handlers

Detailed explanation of your design of the GUI and how the GUI can be used to meet the functionality required by the specification. For example, if you click a button on the GUI, what is going to happen?

Also give necessary screenshot captured when your program is running to demonstrate that output from your work. Describe the properties that you need to set or change for each of the components in your application and clearly show in which event handlers the properties are changed or updated. You can use a table to present the information in an organised manner.

Section 5: Source code

A copy of programme code that contains definition of data members, function members of the classes and all event handlers. Give sufficient comments so a reader can easily understand what you intend to do for each of the variables or event handlers.

Also provide a description at the beginning of each event handler to describe its overall functionality. You can copy the code into a Word document, format the code in an easily read style, and then add text comments to explain the code.

Summary

Does your program work? What test have you done to prove that it works? Does it meet all the requirements? What difficulties you have had in the design and coding stage? If you did not finish the project, what is the reason?