

By Chris Clash

Candidate Number 8070

Table of Contents

[Analysis 2](#_Toc31619533)

[Background 2](#_Toc31619534)

[End User Interview 3](#_Toc31619535)

[General Overview and Objectives of project 4](#_Toc31619536)

[Acceptable Limitations 7](#_Toc31619537)

[Documented Design 8](#_Toc31619538)

[System Design (Overall) – IPSO Chart 8](#_Toc31619539)

[Inputs 8](#_Toc31619540)

[Processes 8](#_Toc31619541)

[Storage 8](#_Toc31619542)

[Outputs 8](#_Toc31619543)

[Modular System Structure 9](#_Toc31619544)

[Record Structures 9](#_Toc31619545)

[Data Requirements 9](#_Toc31619546)

[Validation that will be required 10](#_Toc31619547)

[Description of the game processes 10](#_Toc31619548)

[Initialisation 10](#_Toc31619549)

[Main Game 10](#_Toc31619550)

[Scoreboard + Game Over 11](#_Toc31619551)

[User Interface Design – Main Game and Game Over 12](#_Toc31619552)

[User Interface Design - Scoreboard 13](#_Toc31619553)

[Algorithms 14](#_Toc31619554)

[Asteroid Create Procedure 14](#_Toc31619555)

[Collision Checker + Set Values + Collision Handler 15](#_Toc31619556)

[Move Asteroid 16](#_Toc31619557)

[Appendix A - Art Assets 17](#_Toc31619558)

[Title Art Asset 17](#_Toc31619559)

[Spaceship Asset 17](#_Toc31619560)

[Background Asset 17](#_Toc31619561)

[Asteroid Assets 18](#_Toc31619562)

[Appendix B – Program Code 19](#_Toc31619563)

[UMainGame 19](#_Toc31619564)

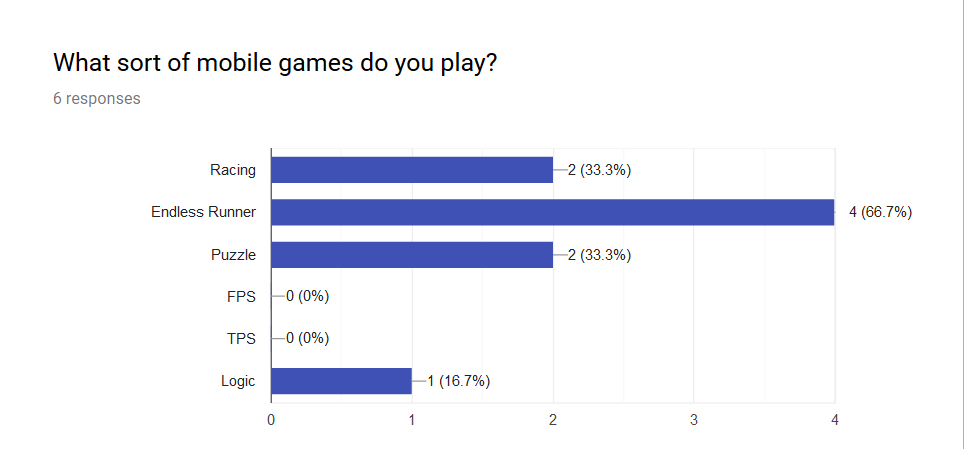
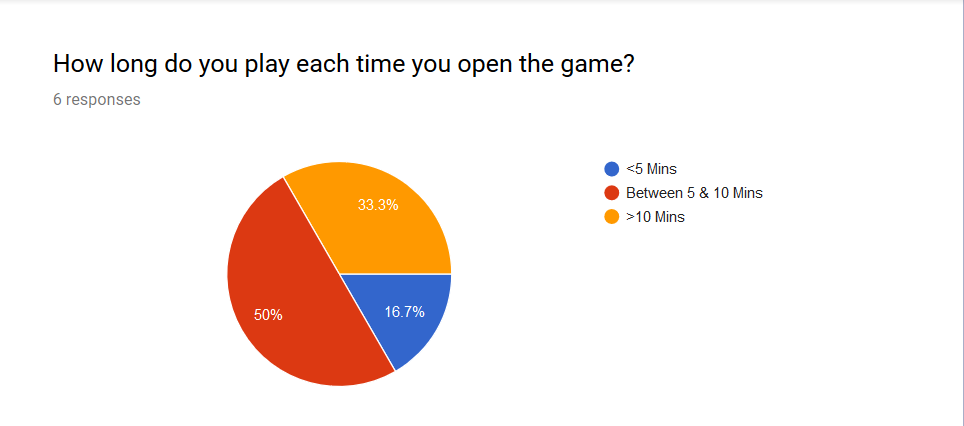
[UGameOver 30](#_Toc31619565)

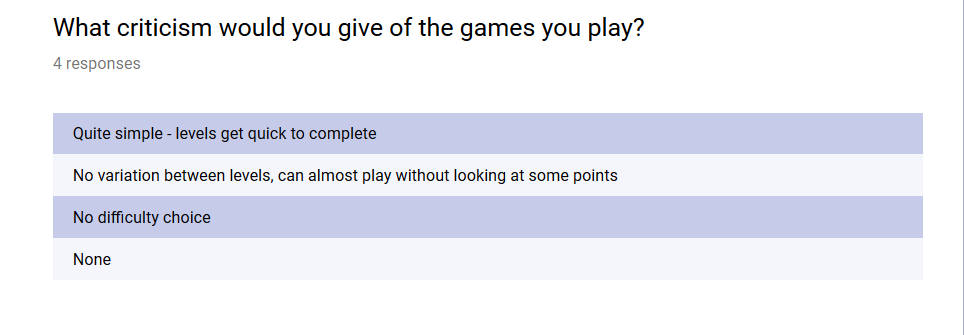
[UScoreBoard 32](#_Toc31619566)

# Analysis

# Background

As smartphone and laptop use has exponentially increased during the last decade, so has the market for simplistic games that are quick and easy (to a degree) to play, most often by people whilst waiting for a short period of time, e.g. waiting for the bus.

After speaking to friends about what they look for in deciding which of these games to play, I decided to put together a very short Google Form Survey [[[1]](#footnote-1)] for them to answer, so as to get an idea as to if there is any consensus on what sort of games they play, and what the unique features of these games are that drew them in. I have included screenshots of the responses to these questions below:



After looking at the results of the survey, I’ve decided to make an Endless Runner game, as this was the most popular option chosen. This is a genre of game where there is no specific ending, instead the game keeps going until the player either makes a mistake and loses, such as hitting an obstacle, or closes the game. The game works by scrolling the background behind/below the player sprite, depending on perspective, so as to create the illusion of movement, the player can then move directionally so as to avoid oncoming obstacles.

I also noticed that the most frequent complaint was that the games were too simplistic, which I will aim to rectify in my project. To determine what should be improved more specifically, I conducted an end-user interview with one of the responders to the survey.

# End User Interview

*Text written like this denotes questions being asked by the interviewer*

Text written like this denotes answers being given by the interviewee

**Text written like this has been added afterwards to explain terminology**

So, what prompted you to ask for this project to be made?

I like to play games on my laptop and phone whilst travelling to and from college but find that games are often either too hard or too easy, so I tend to either get bored or give up. I’d quite like a game that gets more difficult as I get further into it.

Do you prefer certain theming over others?

Yes, I like a good space themed game. I did GCSE Astronomy and it really got me into the whole space scene.

So how would you want the game to be made more difficult the longer you played?

I thought that maybe the ship could go faster, or there could be more obstacles to avoid. This would mean you could just make the proportion of time to the number of obstacles direct, instead of making it random, like most games are at the moment.

*So, what other features would you look for, apart from scaled difficulty (****Game gets progressively harder****)*

I would really like to have some sort of way of seeing other people’s high scores. Most of the games I play tell me that I’ve beaten my own high score, but that doesn’t interest me nearly as much as seeing how I am doing compared to other people.

# General Overview and Objectives of project

I have decided to produce an endless runner, space themed game for my coursework. It will feature a player-controlled ship that moves through a starfield whilst having to dodge asteroids that move faster and faster and appear quicker and quicker the longer the player manages to not hit them. This is to counter the seemingly common problem people had with other games of this type – lack of difficulty.

The game will also feature a global leader board system, so players can set a unique username and then the game will save their highest score, overwriting any previous highscore. At the end of each game, the player will be shown their score from that round, their highscore and its placing globally, as well as the global highscore, so as to add a sense of competition between players.

After the player has survived for a certain length of time, a new enemy ship sprite will be allowed to appear, fire a homing projectile at the player, and then disappear again for an undefined amount of time. This adds an element of surprise to the player, as they will not be able to predict the enemy’s location.

To be able to dodge asteroids and these missiles, the user will be able to activate a time-limited shield using the space key. This will have the effect of disabling the collision detection system for around 5 seconds. As a visual indicator to the user that the shield is active, there will be a coloured ring around the ship that blinks quicker and quicker as the time limit for the shield runs out.

**To make my objectives clearer I have used ‘Debris’ in place of ‘Player Obstacle Entity’ and ‘Ship’ in place of ‘User Controlled Entity’. The terms ‘Player’ and ‘User’ have been used interchangeably, their meaning within context should be assumed to be the same.**

Inputs

1. The program must be able to accept keyboard inputs as controls
2. The program must be able to determine between independent keys, and each key must have a specific function (e.g. D or Right Arrow Key moves the ship to the right)
3. The program must also be able to ignore irrelevant key presses, such as the user pressing a different key to the ones prompted
4. The only keys the game should accept as inputs are the directional arrow keys, WASD and the space bar, menus can be navigated using the mouse and clicking
5. The program must have a start screen, which explains to users the controls and how to play
6. The start screen must be returnable to (can exit the game to the start screen)
7. The program must also have a save score feature, so as to encourage the player to try again (beat the high score).
8. To be able to save the player’s highscore, there must be database that stores the user’s name and highscore
9. The program, when loaded, should prompt the user to enter a name they wish to use
10. The program should then search for the name within the database, and then prompt the user to try a different name if the name they have chosen is already taken (This is a reduced system to make testing within a classroom environment easier, a general release would use a form of autologin, commonplace on most apps (E.g. Google login))
11. The program should then save this allowed name into the database
12. At the end of a round, the program should then be able to search for the highest score within the database, and return that score, along with the associated user’s name on the end splash screen

Processing & Output (Most of the processing directly outputs so it makes sense to merge these into one area)

1. The program must be able to draw the entities in the correct positions
2. To make sure that the entities are always drawn proportionally to each other, their scaling will depend on the resolution of the display being used
3. This will require the program to be able to detect the monitor resolution, and be able to either operate full screen, or take the height of the windows taskbar into consideration when scaling
4. The program must be able to tell the positions of each entity at ALL times, to avoid clipping
5. This will require the program to be able to keep track of each individual entity’s X&Y matrix
6. This will also require the program to be continuously checking whether two matrices are in the same location, and act accordingly if they are
7. To give the program time to identify clipping between pieces of debris, the positions of Debris will be determined, and the objects set to invisible before correct placement is confirmed
8. Once correct placement is confirmed the Debris becomes visible, to maintain fairness and to make the program look aesthetically better, this MUST happen before the Debris is on-screen
9. The program must be able to correctly and smoothly scroll as the player successfully navigates obstacles
10. To reduce the amount of backgrounds required, I have decided to use the same background for the whole game, but rotate the image and flip it to create the effect that the backdrop keeps changing, therefore the program must be able to do this
11. The program must be able to load the modified image before the current one being shown finishes scrolling, so as to maintain a smooth transition between images
12. The program must be able to accurately and quickly detect if there has been a collision between the Ship and a piece of Debris.
13. This requires the program to know the X&Y co-ordinates of every single point along a piece of debris that can interact with the Ship.
14. This is made easier by how Delphi behaves with images, every image is either a square or rectangular shape, with blank space being invisible. This means that the image area can be used to determine a ‘hitbox’ area for the Debris image fairly easily, as the area occupied by this ‘hitbox’ can be found more readily than that of an irregular dodecahedron, for example.
15. The program must be able to constantly keep a track of the area being occupied by the Debris and update the co-ordinate matrix constantly as the Debris moves across the playable area, as this is critical to correct collision-detection.
16. If the Ship co-ordinates match any in the area of a piece of Debris, then this must be detected and flagged for a reaction
17. The program must react by showing a ‘You lose’ (oe.) screen, with either text or a graphic showing that the player has hit Debris
18. The Debris must move in the direction of the player and accelerate as time goes on
19. To make the debris move, the X or Y value will be reduced by a specified amount within a repeating cycle (e.g. once a second)
20. To make the program much simpler to design and get functionally working, Debris will not have collision detection for collisions with other pieces of debris, instead, the approaching piece will be sent behind the other, making it look as if they have simply passed by each other. If I have enough time at the end of the project, I will look into making Debris bounce off of each other
21. As the length of time play has gone on for increases, the speed of the debris should increase, this can be achieved by making the background scroll exponentially faster as the game goes on
22. The Debris generation rate must increase as time goes on, but be limited so the player can still navigate fairly
23. As time in the game goes on, a counter for how many pieces of debris should be on screen should increment by 4, e.g. an extra 4 pieces of debris can appear after every 15 seconds.
24. At a certain limit however, e.g. 50 pieces, the amount of debris on the screen must cap-out, as otherwise the amount would eventually make the screen unnavigable by the player, which is unfair and would reduce the amount of fun in the game
25. After this point a new sprite will be allowed to appear – enemy ships

# Acceptable Limitations

The temptation to bite off more than you can chew when developing a game is infectious, creativity can quickly outweigh logical thinking about what you can achieve within a limited timeframe with limited prior ability. As a result, I’ve had to limit what sort of things I want in my project.

I’ve decided that to keep things simple, the game will be 2D, not 3D. 2D design is far, far easier to do than 3D design, and far less taxing on the system to boot.

As previously mentioned in this analysis, I have decided to target my program for a single platform to begin with. Developing just for Windows will allow to focus entirely on getting the program to run as best as possible with as much working functionality as possible on one platform, whereas splitting my development time between the Windows and Android versions could mean that neither version ends up working very well.

I’ve also decided not to include LAN connected competitive mode, as the technical challenges faced in setting something like that up, as well as making sure that it’s balanced and fun, are too great for me to be able to fit into my project given the time limitations.

# Documented Design

I am designing a two-dimensional, side-scrolling shooter. The player has to dodge oncoming asteroids. The asteroids are randomly generated off-screen before being moved across the screen from right to left.

# System Design (Overall) – IPSO Chart

|  |  |  |  |
| --- | --- | --- | --- |
| Inputs | Processes | Storage | Outputs |
| Keyboard presses  Scores Text file | Asteroid Generation  Background Scrolling  Ship Movement (respond to keypresses)  Movement of the asteroid objects  Check for collision between asteroids and spaceship  Menu navigation  Read and then sort previous user’s scores | Scores text file  Asteroid objects list  Ship coordinates array | Spaceship & Asteroid positions on the screen  UI (User Interface) elements  Previous user’s scores |

To achieve the main goals of the project as outlined in the introduction, there are several major items that I will need to design. These are:

* UI (User Interface)
* Menu System
* Asteroid Object Generation
* Ship Movement
* Background Movement
* Collision engine (axis-aligned bounding box)

# Modular System Structure

The application will be split into the game-over pages (game-over consists of the game over splash screen, as well as the scoreboard screen) and the main game itself. Each will be its own unit/class. Menus will be able to be navigated through a series of signposted buttons, such as Exit or View Scores. The asteroid objects will all be instances of a class and be managed under the umbrella of the main game class. The ship’s controls will be controlled by the main game class. Background movement, as well as the collision engine, will both be separate classes. Asteroid movement will also be handled by a separate class.

# Record Structures

Almost all of the data I will be using is generated dynamically at run-time. I will be storing a consistently updated array of the Spaceship’s co-ordinates, as well as a List of Asteroid objects that have been created. This list will be updated when a new asteroid is created. I will also be using a generic text file to store user’s scores. Due to the minor role this plays within the larger program, I have opted against using a binary file, as this takes longer to program. Apart from the scores text file, all data stored by the program is stored in intentionally volatile locations, as data, such as the co-ordinates of the spaceship, does not need to be carried between instances of the game running.

# Data Requirements

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Name | Class | Purpose | Type | Example of use |
| CoordinateList | Main Game | Threadsafe list that stores all of an asteroid object’s properties | Threadlist | [AsteroidName, AsteroidParent, AsteroidImage, AsteroidTop, AsteroidLeft] |
| AsteroidList | MoveAsteroidThread | Is used to be able to access the data stored in CoordinateList | List | [AsteroidName, AsteroidParent, AsteroidImage, AsteroidTop, AsteroidLeft] |
| SpaceshipArea | Main Unit | Stores the co-ordinates of every point on the form occupied by the spaceship object | Array of TPoint | [(12,40),(34,40)] |
| ScoreFile | Game Over | Writes the user’s score to a pre-existing text file | TextFile | 40 |

# Validation that will be required

Because of the way that Delphi handles keyboard keypresses during runtime, I will not need to include much in the way of validation for the user pressing the key, as by default Delphi simply ignores these. Instead, I will need to include validation to detect when an anticipated key is pressed.

As my program will be using a pre-existing text file, I will need to ensure that I check that the file already exists before attempting to read/write to it, or else risk my program crashing if it fails to find the text file.

As my program uses the length of a list to perform certain functions, I need to validate whether the list has any items in it, as otherwise this will lead to a crash.

# Description of the game processes

The game can be split into 3 sections: Initialisation, main game & scoreboard + game over. Each of these sections have reactionary or non-reactionary processes.

# Initialisation

This part of the program deals with setting up the game

* Load the backgrounds into the program so that they’re ready to be used
* Position the backgrounds so that they scroll correctly
* Position the spaceship in the starting position
* Prepare the asteroid class ready to be used during the main game sequence
* Initialize all the threads so that they are ready to begin their functions
* Verify that the ScoreSave text file is at the address expected

# Main Game

During the main game sequence of the program, many of the critical threads run indefinitely. This is important for creating aspects of the program like the infinitely scrolling background. Other procedures run in response to a player action, like the player pressing a key on the keyboard.

* Increment the backgrounds to the left
* Create new instantiations of asteroid objects off screen
* If a new asteroid is generated, increment the score by 20
* Move existing asteroid objects towards the player by editing the X and Y co-ordinate aspects of each asteroid object
* Respond to player inputs on the keyboard appropriately (W moves the player upwards; S moves the player down etc.). This is done by modifying the X and Y co-ordinates of the spaceship.
* Parse the co-ordinates of each asteroid, as well as those belonging to the spaceship into the collision engine function.
* The collision engine compares the area on the form occupied by the asteroid being inspected to the area on the form occupied, returning a Boolean value
* If the Boolean value returned by the collision engine is **True**, call the collision handler
* If the collision engine is called, end the main game sequence

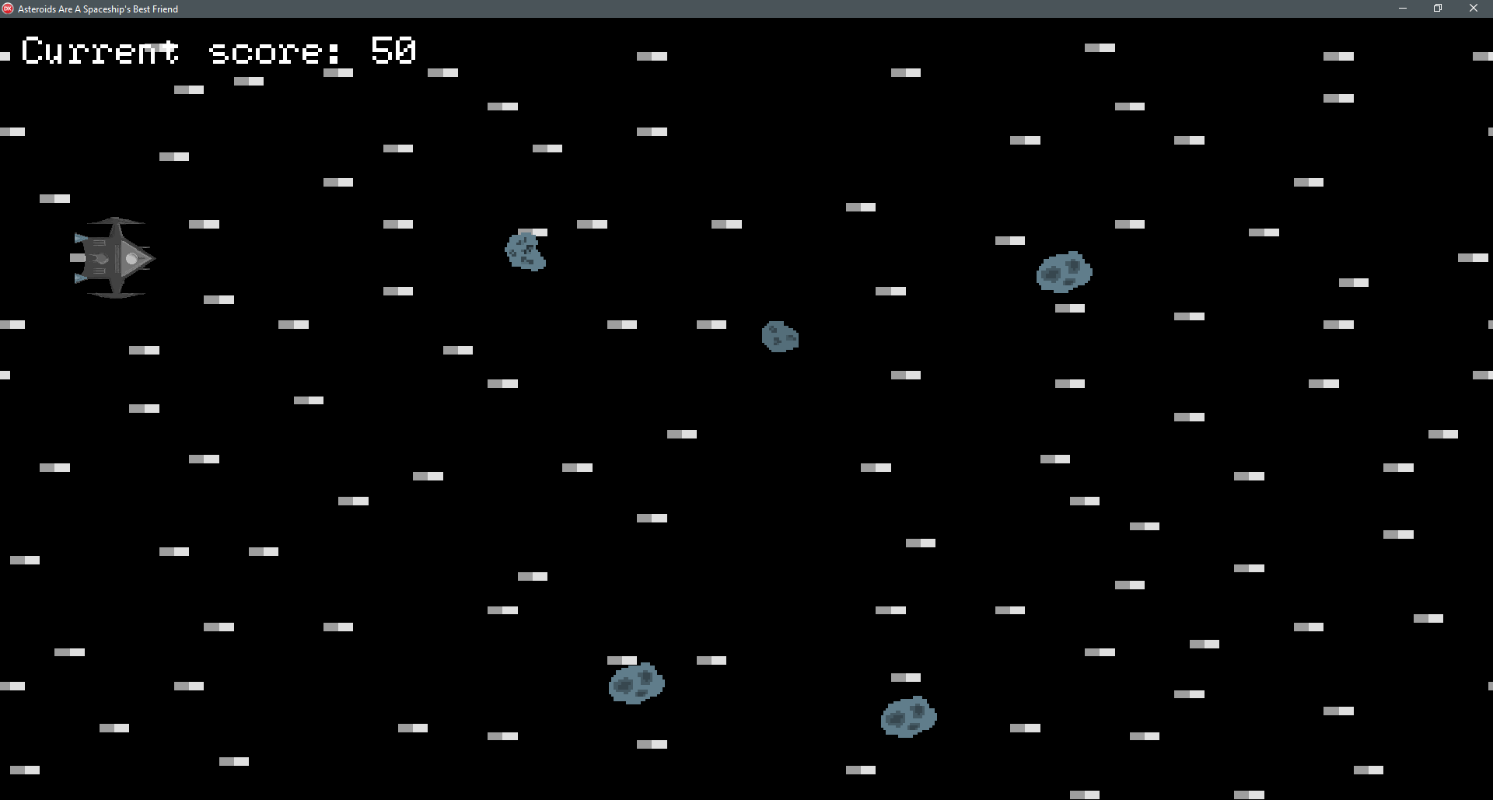
# Scoreboard + Game Over

When the collision handler is called, the game is over, and the following subroutines are called

* Suspend new asteroid generation
* Write the player’s score to the text file
* Show the user two buttons – ‘exit’ or ‘show score’
* Read every value from the ScoreSave file and place them into an array
* Perform a descending bubble sort on this array, populate the scoreboard with the first 5 items in the array
* If the user chooses ‘exit’, close every form. If the user chooses ‘show scores’ then show the scoreboard
* If show score is selected, then hide the game over screen and show the scoreboard

# User Interface Design – Main Game and Game Over

A screen shot of a computer

Description automatically generated

Menu buttons, ‘Exit’ and ‘Show Scores’

Player’s final score

Approaching enemies - Asteroids

Score counter

Player

# User Interface Design - Scoreboard

A screenshot of a cell phone

Description automatically generated

Highest score is highlighted in red

List of players’ highscores – read from text file

# Algorithms

# Asteroid Create Procedure

This algorithm, when called by its owner thread, creates an instance of an asteroid object, defining the critical aspects of the object. The Y co-ordinate is defined as a random integer between 20 and 1000, whilst the X co-ordinate is fixed. This results in the asteroid always generating off the screen, but at varying heights. The algorithm also uses a random number generator to determine which image of three to use as the picture for the asteroid. If the algorithm determines that the new asteroid object will be generated too close to the previous asteroid, it will generate a new Y co-ordinate

*Asteroid Create Procedure*

*Num = random integer between 1 & 3*

*Create Asteroid TImage object*

*Asteroid Parent = Main Game Form*

*Asteroid Name = ‘Asteroid’ + I*

*Asteroid Height & Width = 100,100*

*If Num = 1*

*Asteroid Picture = Picture 1*

*Else if Num = 2*

*Asteroid Picture = Picture 2*

*Else if Num = 3*

*Asteroid Picture = Picture 3*

*Asteroid Y co-ordinate = random integer between 20 & 1000*

*If (YPrevious - Asteroid Y co-ordinate) < 100*

*Generate a new Y co-ordinate*

*Else if (Asteroid Y co-ordinate – YPrevious) < 100*

*Generate a new Y co-ordinate*

*Else*

*Asteroid X co-ordinate = 2000*

*Asteroid Visible = True*

*Bring Asteroid to the front*

*Add Asteroid to the CoordinateList*

*YPrevious = Asteroid Y co-ordinate*

# Collision Checker + Set Values + Collision Handler

These two algorithms work in tandem to perform the most critical role in the whole program. Set Values defines 4 variables as the co-ordinates of the top left and bottom right corners of the rectangle that serves as the Spaceship Sprite before calling Collision Check, parsing the 4 variables that have just been defined. Collision check then populates an array of type TPoint (Delphi’s way of storing co-ordinates) with the co-ordinates of every point on the form being occupied by the Spaceship sprite before using the asteroid corner co-ordinates to create a TRectangle with the same co-ordinates. It then compares every item in the array to the TRectangle, with a match being flagged as a positive for collision. This method of collision checking is called an axis-aligned bounding box engine. If a collision is detected, the collision handler is called.

*Function Set Values*

*SpaceshipXLow = Spaceship Left co-ordinate*

*SpaceshipXHigh = SpaceshipXLow + 105*

*Spaceship YLow = Spaceship Top*

*Spaceship YHigh = SpaceshipYLow + 105*

*Set Values = CollisionCheck(SpaceshipXLow, SpaceshipXHigh, SpaceshipYLow, SpaceshipYHigh)*

*If Set Values = True*

*Call Procedure Collision Handler*

*Function Collision Checker*

*SpaceshipArea = Array [0 to 11025] of type TPoint*

*AsteroidLowX = Asteroid Left*

*AsteroidHighX = Asteroid Left + 100*

*AsteroidLowY = Asteroid Top*

*AsteroidHighY = Asteroid Top + 100*

*I = 0*

*For A = 0 to 105*

*For B = 0 to 105*

*SpaceshipArea[I].X = SpaceshipXLow*

*SpaceshipXLow = SpaceshipXLow + 1*

*SpaceshipArea[I].Y = SpaceshipYLow*

*I = I + 1*

*SpaceshipYLow = SpaceshipYLow + 1*

*For Z = 0 to Length(SpaceshipArea)*

*If PtInRect(AsteroidRect, SpaceshipArea[Z] = True*

*Collision Checker = True*

*Procedure Collision Handler*

*Show Game Over Screen*

*Suspend Asteroid creation*

# Move Asteroid

This algorithm is responsible for moving the asteroids. It firstly checks if the list contains any asteroid items, as trying to act on an empty list would cause the program to crash. When it finds an asteroid object in the list, it changes that asteroid’s X – co-ordinate by 25 to the left. After this, it then parses the new co-ordinates of the asteroid into the collision checking subroutine system. Finally, when an asteroid has moved entirely off of the screen, it is destroyed and then removed from the list.

*Procedure MoveAsteroid*

*AsteroidList = Lock CoordinateList //Must lock threadlist before editing it*

*Try*

*If Count of items in AsteroidList <> 0*

*For I = 0 to (Count of items in AsteroidList – 1)*

*TImage(AsteroidList[I]).Left = TImage(AsteroidList[I]).Left – 25*

*CollisionCheck = (Spaceship Co-ordinates, TImage(AsteroidList[I]).Left, TImage(AsteroidList[I]).Top)*

*If CollisionCheck = True*

*Call the collision handler*

*If TImage(AsteroidList[0].Left) < -225*

*Destroy AsteroidList[0]*

*Delete item 0 from AsteroidList*

*Finally*

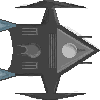
*Unlock Co-ordinate list*

# Appendix A - Art Assets

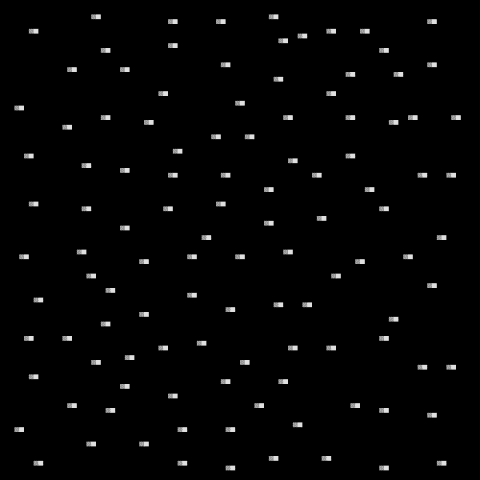
# Title Art Asset



# Spaceship Asset



# Background Asset



The three backgrounds used are all the same image

Asteroid Assets

Asteroid 1 Asteroid 2 Asteroid 3

# Appendix B – Program Code

# UMainGame

unit UMainGame;

interface

uses

Winapi.Windows, Winapi.Messages, System.SysUtils, System.Variants, System.Classes, Vcl.Graphics, Vcl.Controls, Vcl.Forms, Vcl.Dialogs, Vcl.ExtCtrls, Vcl.Imaging.pngimage,

Vcl.Imaging.GIFImg, Data.Bind.EngExt, Vcl.Bind.DBEngExt, System.Rtti,

System.Bindings.Outputs, Vcl.Bind.Editors, Data.Bind.Components, Math, UGameOver;

type

TMainGameForm = class(TForm)

imgShipFrameFocus: TImage;

imgFrame1: TImage;

imgFrame2: TImage;

imgFrame3: TImage;

imgGameBackground: TImage;

imgGameBackground2Frame: TImage;

ThreadStart: TTimer;

imgStaticBackground: TImage;

CollisionCheck: TTimer;

procedure ThreadStartTimer(Sender: TObject);

procedure FormKeyPress(Sender: TObject; var Key: Char);

procedure CollisionHandler;

public

Score : integer;

end;

var

SpaceshipXHigh, SpaceshipXLow, SpaceshipYLow, SpaceshipYHigh, AsteroidLeft, AsteroidTop, IncrNum, AsteroidNum: integer;

MainGameForm: TMainGameForm;

Collision : Boolean;

BackgroundScrollThread: TThread;

ShipThread : TThread;

CreateAsteroidThread : TThread;

MoveAsteroidThread : TThread;

Asteroid : TImage;

CoordinateList : TThreadList;

function SetValues(SpaceshipXHigh, SpaceshipXLow, SpaceshipYHigh, SpaceshipYLow, AsteroidLeft, AsteroidTop : integer) : boolean;

function CollisionCheck(SpaceshipXLow, SpaceshipYLow, SpaceshipXHigh, SpaceshipYHigh, AsteroidLeft, AsteroidTop : integer) : boolean;

implementation

{$R \*.dfm}

type TBackgroundScrollThread = class(TThread) //Defines the BackgroundScrollThread methods

public

procedure Execute; override;

procedure MoveScreen;

end;

procedure TBackgroundScrollThread.Execute;

var Run : boolean;

begin

Run := True;

while Run = True do

begin

Sleep(1);

Synchronize(MoveScreen);

end;

end;

procedure TBackgroundScrollThread.MoveScreen;

begin

MainGameForm.imgGameBackground.Left := MainGameForm.imgGameBackground.Left - 10;

MainGameForm.imgGameBackground2Frame.Left := MainGameForm.imgGameBackground2Frame.Left - 10;

if MainGameForm.imgGameBackground.Left < -1920 then

begin

MainGameForm.imgGameBackground.Left := 0;

end;

if (MainGameForm.imgGameBackground2Frame.Left < 0) then

Begin

MainGameForm.imgGameBackground2Frame.Left := 1920;

End;

end;

type TShipThread = class(TThread)

var IncrNum : integer;

public

procedure Execute; Override;

procedure IncrImage;

end;

procedure TShipThread.Execute;

var Run : Boolean;

begin

Run := True;

IncrNum := 1;

while Run = True do

begin

Synchronize(IncrImage);

IncrNum := IncrNum + 1;

Sleep(80);

end;

end;

procedure TShipThread.IncrImage;

var Run : Boolean;

begin

begin

// if IncrNum = 1 then

// begin

// MainGameForm.imgShipFrameFocus.Picture := MainGameForm.imgFrame1.Picture;

// end

// else if IncrNum = 2 then

// begin

// MainGameForm.imgShipFrameFocus.Picture := MainGameForm.imgFrame2.Picture;

// end

// else if IncrNum = 3 then

// begin

// MainGameForm.imgShipFrameFocus.Picture := MainGameForm.imgFrame3.Picture;

// IncrNum := 1; //Resets back to picture 1

//end

end;

end;

type TMoveAsteroidThread = class(TThread)

public

procedure execute; override;

procedure MoveAsteroids;

end;

procedure TMoveAsteroidThread.MoveAsteroids;

var AsteroidList : TList;

var I : integer;

var CollisionCheck : boolean;

begin

AsteroidList := CoordinateList.LockList;

try

if AsteroidList.Count <> 0 then

begin

for I := 0 to AsteroidList.Count - 1 do

begin

TImage(AsteroidList.Items[I]).Left := TImage(AsteroidList.Items[I]).Left - 25;

CollisionCheck := UMainGame.SetValues(SpaceshipXHigh, SpaceshipXLow, SpaceshipYHigh, SpaceshipYLow, TImage(AsteroidList[I]).Left, TImage(AsteroidList[I]).Top);

if CollisionCheck = True then

begin

MainGameForm.CollisionHandler;

end;

end;

if TImage(AsteroidList.Items[0]).Left < -225 then

begin

TImage(AsteroidList.Items[0]).Destroy;

AsteroidList.Delete(0);

end;

end;

finally

begin

CoordinateList.UnlockList;

end;

end;

end;

procedure TMoveAsteroidThread.Execute;

var Run : boolean;

begin

Run := True;

while Run = True do

begin

sleep(1);

synchronize(MoveAsteroids);

end;

end;

type TCreateAsteroidThread = class(TThread)

var Asteroid : TImage;

var YCoord, XCoord, YPrevious, Num : integer;

var AsteroidList : TList;

public

procedure Execute; Override;

procedure CreateAsteroid;

end;

procedure TCreateAsteroidThread.CreateAsteroid;

begin

Num := RandomRange(1,4);

Asteroid := TImage.Create(Nil);

Asteroid.Parent := MainGameForm;

Asteroid.Name := 'Asteroid' + inttostr(AsteroidNum);

Asteroid.Visible := False;

Asteroid.Height := 100;

Asteroid.Width := 100;

if Num = 1 then

begin

Asteroid.Picture.LoadFromFile('E:\CompSci Final Release\Asteroid Sprites\Asteroid1.png')

end

else if Num = 2 then

begin

Asteroid.Picture.LoadFromFile('E:\CompSci Final Release\Asteroid Sprites\Asteroid2.png')

end

else if Num = 3 then

begin

Asteroid.Picture.LoadFromFile('E:\CompSci Final Release\Asteroid Sprites\Asteroid3.png')

end;

YCoord := RandomRange(20,1000);

if ((YPrevious - YCoord) < 100) then

begin

YCoord := RandomRange(100,1000);

end

else if ((YCoord - YPrevious) < 100) then

begin

YCoord := RandomRange(20,1000);

end

else

begin

Asteroid.Top := YCoord;

Asteroid.Stretch := True;

Asteroid.Left := XCoord;

Asteroid.Visible := True;

Asteroid.BringToFront;

CoordinateList.Add(Asteroid);

YPrevious := YCoord;

end;

end;

procedure TCreateAsteroidThread.Execute;

var Run : boolean;

var I : integer;

begin

AsteroidNum := 3;

Run := True;

XCoord := 1920;

while Run = True do

begin

Sleep(1750);

XCoord := XCoord + 150;

for I := 0 to (AsteroidNum + 1) do

begin

sleep(500);

YPrevious := 1000;

CreateAsteroid;

AsteroidNum := AsteroidNum + 1;

MainGameForm.Score := MainGameForm.Score + 10;

end;

end;

end;

function CollisionCheck(SpaceshipXLow, SpaceshipYLow, SpaceshipXHigh, SpaceshipYHigh, AsteroidLeft, AsteroidTop : integer) : boolean;

var SpaceShipArea : Array[0..11025] of TPoint;

var Xa, Ya, A, B, I, Z, AsteroidHighX, AsteroidLowX, AsteroidHighY, AsteroidLowY : integer;

var AsteroidRect : TRect;

begin

Xa := SpaceshipXlow;

Ya := SpaceshipYLow;

begin

AsteroidLowX := AsteroidLeft;

AsteroidHighX := AsteroidLowX + 100;

AsteroidLowY := AsteroidTop;

AsteroidHighY := AsteroidLowY + 100; object

AsteroidRect := Rect(AsteroidLowX, AsteroidLowY, AsteroidHighX, AsteroidHighY);

I := 0;

for A := 0 to 105 do

begin

for B := 0 to 105 do

begin

SpaceShipArea[I].X := SpaceshipXLow;

SpaceshipXLow := SpaceshipXLow + 1;

SpaceshipArea[I].Y := SpaceshipYLow;

I := I + 1;

end;

SpaceshipYLow:= SpaceshipYLow + 1;

SpaceshipXLow := Xa;

end;

SpaceshipYLow := Ya;

A := 0;

for Z := 0 to length(SpaceshipArea) do

begin

if PtInRect(AsteroidRect,SpaceshipArea[Z]) = True then

begin

CollisionCheck := True;

Break

end;

end;

end;

end;

function SetValues(SpaceshipXHigh, SpaceshipXLow, SpaceshipYHigh, SpaceshipYLow, AsteroidLeft, AsteroidTop : integer) : boolean;

var Check : boolean;

begin

SpaceshipXLow := MainGameForm.imgShipFrameFocus.Left;

SpaceshipXHigh := (SpaceshipXLow + 105);

SpaceshipYLow := MainGameForm.imgShipFrameFocus.Top;

SpaceshipYHigh := SpaceshipYLow + 105;

Check := CollisionCheck(SpaceshipXLow, SpaceshipYLow, SpaceshipXHigh, SpaceshipYHigh, AsteroidLeft, AsteroidTop);

if Check = True then

begin

SetValues := True

end

else SetValues := False;

end;

procedure TMainGameForm.CollisionHandler;

begin

GameOverScreen.Show;

end;

procedure TMainGameForm.FormKeyPress(Sender: TObject; var Key: Char);

var ProxToTB, ProxToLR : integer;

begin

ProxToTB := MainGameForm.imgShipFrameFocus.Top;

ProxToLR := MainGameForm.imgShipFrameFocus.Left;

if (Key in ['w', 'W']) and (ProxToTB > -10) then

begin

MainGameForm.imgShipFrameFocus.Top := MainGameForm.imgShipFrameFocus.Top - 15;

end;

if (Key in ['s', 'S']) and (ProxToTB < 1050) then

begin

MainGameForm.imgShipFrameFocus.Top := MainGameForm.imgShipFrameFocus.Top + 15;

end;

if (Key in ['d', 'D']) and (ProxTOLR < 1920)then

begin

MainGameForm.imgShipFrameFocus.Left := MainGameForm.imgShipFrameFocus.Left + 15;

end;

if (Key in ['a', 'A']) and (ProxToLR > -20) then

begin

MainGameForm.imgShipFrameFocus.Left := MainGameForm.imgShipFrameFocus.Left - 15;

end;

end;

procedure TMainGameForm.ThreadStartTimer(Sender: TObject);

.begin

imgStaticBackground.SendToBack;

BackgroundScrollThread := TBackgroundScrollThread.Create(false);

ShipThread := TShipThread.Create(false);

CreateAsteroidThread := TCreateAsteroidThread.Create(false);

MoveAsteroidThread := TMoveAsteroidThread.Create(false);

ThreadStart.Enabled := False;

end;

begin

IncrNum := 1; //Sets IncrNum to 1

CoordinateList := TThreadList.Create;

# UGameOver

unit UGameOver;

interface

uses

Winapi.Windows, Winapi.Messages, System.SysUtils, System.Variants, System.Classes, Vcl.Graphics,

Vcl.Controls, Vcl.Forms, Vcl.Dialogs, Vcl.StdCtrls, Vcl.ExtCtrls,

Vcl.Imaging.jpeg, UScoreBoard;

type

TGameOverScreen = class(TForm)

ExitGame: TButton;

Panel1: TPanel;

Logo: TImage;

GameOverMessage: TLabel;

ViewScores: TButton;

PlayerScore: TLabel;

procedure ExitGameClick(Sender: TObject);

procedure FormShow(Sender: TObject);

procedure ViewScoresClick(Sender: TObject);

procedure FormClose(Sender: TObject; var Action: TCloseAction);

end;

var

GameOverScreen: TGameOverScreen;

CloseGame : boolean;

EndScore : integer;

ScoreFile : TextFile;

Scores : string;

implementation

{$R \*.dfm}

uses UMainGame;

procedure TGameOverScreen.ExitGameClick(Sender: TObject);

begin

MainGameForm.Close

end;

procedure TGameOverScreen.FormClose(Sender: TObject; var Action: TCloseAction);

begin

MainGameForm.Close

end;

procedure TGameOverScreen.FormShow(Sender: TObject);

begin

EndScore := MainGameForm.Score;

AssignFile(ScoreFile, 'E:\CompSci Final Release\Scores.txt');

Append(ScoreFile);

Writeln(ScoreFile, EndScore);

CloseFile(ScoreFile);

PlayerScore.Caption := 'Your score was: ' + inttostr(EndScore);

end;

procedure TGameOverScreen.ViewScoresClick(Sender: TObject);

begin

ScoreBoard.Show;

end;

end.

# UScoreBoard

unit UScoreBoard;

interface

uses

Winapi.Windows, Winapi.Messages, System.SysUtils, System.Variants, System.Classes, Vcl.Graphics,

Vcl.Controls, Vcl.Forms, Vcl.Dialogs, Vcl.ExtCtrls, Vcl.StdCtrls;

type

TScoreboard = class(TForm)

Score1: TLabel;

Score2: TLabel;

Score3: TLabel;

Score4: TLabel;

Score5: TLabel;

HighscoreLabel: TLabel;

procedure FormShow(Sender: TObject);

end;

var

Scoreboard: TScoreboard;

HighScores : TextFile;

Scores : integer;

implementation

uses UGameOver, UMainGame;

{$R \*.dfm}

procedure TScoreboard.FormShow(Sender: TObject);

var i, x, y, Temp : integer;

var ScoreArray : array[0..50] of integer;

begin

AssignFile(HighScores, 'E:\CompSci Final Release\Scores.txt');

Reset(Highscores);

while not Eof(HighScores) do

begin

for i := 0 to 50 do

begin

readln(Highscores, Scores);

ScoreArray[i] := Scores;

end;

end;

CloseFile(HighScores);

for x := 0 to 49 do

begin

for y := x + 1 to 50 do

begin

if ScoreArray[x] < ScoreArray[y] then

begin

Temp := ScoreArray[x]; elsewhere

ScoreArray[x] := ScoreArray[y];

ScoreArray[y] := Temp;

end;

end;

end;

Score1.Caption := inttostr(ScoreArray[1]);

Score2.Caption := inttostr(ScoreArray[2]);

Score3.Caption := inttostr(ScoreArray[3]);

Score4.Caption := inttostr(ScoreArray[4]);

Score5.Caption := inttostr(ScoreArray[5]);

end;

end.

1. <https://forms.gle/pKn9P8Y1Thh21xtK8>

   All screenshots, including on page 3, taken from same source [↑](#footnote-ref-1)