Android Game for Measurement of Perceived Control

Bachelor of Science in Cyber Security and IT Forensics

Final Year Project
Final Report

Ian Rowland
19190859

Pepijn Van De Ven

15/03/2023

Abstract

The concept of perceived control refers to the belief of an individual as to their own ability to influence both their internal state and behaviours, as well as influence their external environment.

The motivation for this project was to develop a simple and fun android game that could unobtrusively , and quantitatively aid in the assessment of a user’s level of perceived control, both by posing a questionnaire to the user after their gameplay session has concluded, and by measuring a set of factors associated with a user’s inputs/taps on the screen, such as pressure that is applied to the screen, the duration of the inputs, the time between inputs, and accelerometer values for the relevant android device. The application will, on occasion, intentionally not follow a user’s input, in order to measure how a user responds to such an event, in comparison to an ordinary input.

Throughout the course of this project, the intended android application was developed, with the majority of the intended functionality for assessing a user’s level of perceived control having been implemented.

In addition, a python notebook was also developed, which allows for the data gathered by the application to be stored and visualised in different types of graphs, such as line graphs and scatterplots, to allow for direct comparisons between ordinary inputs, and inputs that follow an input that was not followed by the game.

This report will discuss the technical theory behind the project, as well as the specification, design and implemented features within the application, as well as the results of the project work, and how the gathered data may be visualised.

Declaration

This report is presented in part fulfilment of the requirements for the Bachelor of Science in Cyber Security and IT Forensics **Final Year Project**.

It is entirely my own work and has not been submitted to any other University or Higher Education Institution or for any other academic award within the University of Limerick.

Where there has been made use of work of other people it has been fully acknowledged and referenced.

Name	**Ian Rowland**
Signature	Ian Rowland
Date	**15/03/2023**

Table of Contents

List of Figures

List of Tables

Chapter 1 Introduction

The objective of this project was to develop a simple and fun game for the android operating system, that will unobtrusively test a users level of perceived control, by randomly and deliberately not following a users inputs at certain points throughout a game session.

The game that has been developed for this project is a simple 2D endless scrolling game that is inspired by the flappy bird mobile game. The base premise of this game is for the user to control a bird that is flying through the sky, and trying to avoid two types of differently coloured clouds, and aim for 1 type of cloud. These cloud types include blue clouds, which inflict one point of damage to the player, red clouds, which inflict two points of damage, and green clouds, which restore one point of health.

The perceived control would be measured by measuring different metrics related to a users inputs, such as the time between a users inputs, measured from when the previous input ends to when the new input starts, the duration of an input, the pressure applied to the screen by the user, and device values related to the device accelerometer upon input.

Following the main gameplay, a short questionnaire is posed to the user to help assess the users level of perceived control within the game. These answers are then saved to a database, and the raw input metrics are saved to a text file with an id number associated with the user's session entry id within the database, i.e a user with an id of 7 will also produce and inputs text file labelled as "inputs7.txt". This will allow the data to be reviewed and interpreted appropriately outside of the app, such as by means of a python notebook.

Chapter 2 Background Theory

This section will discuss the core background theory behind perceived control that served as a motivation for the early stages of the project, and will detail some of the key concepts of perceived control as relevant to the project.

The core background theory for this project relates to the concepts of perceived control. Perceived control refers to the belief of an individual as to their own ability to influence both their internal states and behaviors, as well as on their external environment [1].

Seligman’s research on perceived control described that the perceived control of a situation resulted in a specific outcome of behavior. This research led to Seligman coining the term ‘learned helplessness’ with respect to perceived control. This suggests that, when presented with a situation where an event occurs beyond someone’s control, that individual will learn to accept that situation, and not attempt to resist it [2]

This concept of learned helplessness was a concept that greatly intrigued me for much of the time that was spent working on this project. Based on early research conducted at the start of this project, I hypothesised that there may be the possibility that, in an event where a user is experiencing repeated incidents of inputs not being followed, that the nature of their reactions to their inputs not being followed may change with repeated tests for perceived control, whether that be an increase in intensity of certain reactions, or a general change in behaviour that may arise from a user realising that certain inputs are not being followed.

In the context of a game, there are four types of control. These types of control can be described as: Fully Controlled, Probabilistic Control, Emergent Control and Uncontrolled. Thes different types of control, as their names suggest, refer to different levels of control that an individual may have in the context of a game, whether that game is played by means of a computer, or by non-technological means.[3]

Fully Controlled:

In a fully controlled system, all outcomes of all actions are predetermined, and the participant in such a system will have an understanding of all of these outcomes. The result of such a system is that there are no true stakes to speak of, so such a fully controlled system will not typically be considered to be a true “game” [3].

Emergent Control:

In an emergent control-based system, and individual will typically operate within a defined set of rules, where the behaviour and outcomes of the system are unpredictable, in ways that cannot be calculated. The individual is then able to respond to the behaviours of the system accordingly, potentially based on how they believe the system will react. This individual, while not actually in control of the game, is able to make choices based on the behaviour and outcomes of such a system. As a result of this, the individuals may be able to perceive a certain degree of control, while not actually being in true control of the system. In the context of a game, particularly a video game, this feeling of control contributes to the sensation of “being in the zone”, which can be described, in a more technical fashion, as the flow of a game. This level of control may provide an individual with some ability to exert a level of control over a system, provided that the individual is able to gain an understanding of the underlying model of the system, and/or how that system works at a sufficient level.[3]

Probabilistic control:

Probabilistic control refers to a type of control where the participant will be aware of the uncertainty of a game, or the uncontrollable nature of certain aspects of it, prior to their participation in that game, in a fashion that is either intuitive or mathematical in nature. As an example, a game that is based on rolling some dice, or drawing cards from a randomly assorted pile, where the player can determine the probability of particular results occurring.[3]

According to Beth Arburn Davis' paper "Development and Validation of a Scale of Perceived Control Across Multiple Domains", the best way to measure perceived control involves asking questions either directly about a person's ability to perform a certain behaviour, or indirectly regarding a persons beliefs about their ability to deal with certain factors that either inhibiting or facilitating in nature [4].

This fact regarding best practices for assessing perceived control has influenced the questions that are to be posed to the user of the developed application, as these questions are concerned with asking a player direct questions regarding how much control they believed themselves to have, the level of control they believe the system had, and whether or not they felt themselves losing control of the system over time, or whether they gained control over time. Another question that is posed asks about whether not a user believed that they were able to regain control in situations where they felt they lost control of the system. Such a question would be asked to assess if a user feels that they have a level of influence over the system in which they are participating.

The main questions that are posed to users are as follows:

Did you enjoy the game?

How much control did you feel you had over the game?

Did you feel like your control increased or decreased over time?

How much control did you feel the game/system had over your actions?

In situations where you feel that you lost control, did you feel like you were able to regain control?

Chapter 3 Declaration of Code Resources

A number of programming-based resources were used to aid in the development of this project. These resources were largely centred around the development of the GameView/SurfaceView class that was used to define key gameplay operations.

The majority of these sources were tutorials on android game development, which offered an introduction into the core operations that would be used for a game such as the one developed for this project. Some of these sources offered assistance in using threads to enable smooth gameplay operations.

One of these sources was a basic introduction to android game development [5]. This tutorial provided a brief introduction to some of the technical components required to make an android game.

Another of these resources was a guide/tutorial on how to make a basic flappy bird-like game in android studio, which helped to introduce the use of threads and classes defining particular characters and obstacles on the screen to this project[6].

The flying fish game project was used as an initial source for learning how to use and extend SurfaceViews for the GameView class. The Intent.putExtras and Intent.getExtras methods that were used to allow for the passage of variables between activities was also learned from this project [7].

Chapter 4 Technical Theory

As the application was developed using android studio, the Java programming language was used for the development of the application.

Activity classes are used within the developed application to define the actions and layouts associated with each screen within the application. Each activity represents one screen and the actions associated with it. The main activity, which is the activity that is started upon the opening of the application, allows for access to other activities via the use of Intents [8].

Intents are used to move from one activity to another within the application. These intents are generally triggered by an appropriate button being pressed, except in the case of the gameplay activity and surfaceView, where the intent to move to the next activity is triggered automatically upon the game being finished [9].

Additionally, these intents can also be used to allow variables to be moved from one activity to another for use in a subsequent activity as needed. This can be achieved with the "Intent.putExtras()" method, which takes a 'nametag' of sorts that can be used to identify the variable to be called in the next activity, as well as the variable itself. The main use for this allows the score that is achieved for the game, which is defined in the surfaceView class, to be transferred to and read from the gameOverActivity[7].

In terms of measuring the users' inputs within the application, the motionevent class is used to determine the pressure that the user is applying to the screen for each input. It is also used to determine whether a user has touched the screen, thereby starting an input event, or if they have just removed their finger from the screen to end an input [10].

The overall game details and rules are defined within a surfaceview class, that allows for the player character and obstacle objects to be drawn onto an on-screen canvas, with their positioning on the screen being updated upon the x and y co-ordinates of each object is updated within the code. This class is, for all intents and purposes, an alternative means of defining the layout and general appearance of an activity, as opposed to using the traditional xml layout files, which are more appropriate for use in activities that have a static layout that does not naturally change over time [11].

This surfaceView class contains the methods associated with updating the view, such as methods that allow the player character and obstacle images to be drawn to the screen, methods that define the game rules and runtime behaviour of the gameplay, and a method for ending the game and moving to the next activity upon the relevant conditions being met.

The classes associated with the player character and the obstacles within the game are visually represented on the gameplay screen by bitmap objects which are contained within the relevant character and obstacle classes.

Threads are also used in this application to aid in the execution of the operations associated with core gameplay. By default, code in an android application is executed on the systems main UI thread. However, code that is executed over a long period of time may result in performance issues, and as such, would need to be relegated to a separate thread to ensure smooth performance [5].

The accelerometer values are obtained through the implementation of the accelerometer sensor within the surfaceView class. This allows the device acceleration at the time of an input to be recorded a stored appropriately during an input event. Additionally, the accelerometer axis values can also be recorded, with the x axis, y axis and z axis values being recorded, so that the accelerometer data can be interpreted and calculated separately [12].

File input and output functionality is also used within the application to aid in the recording of user input data. While a database would allow for effective storage of input data, this data would, for the purposes of this project, need to be viewable and visualised outside of the developed application, and thus, file input and output was deemed to be the most appropriate means of recording input data and questionnaire answers.

Despite this, an internal application database is used for the purposes of recording an individual users answers to the questionnaire that is posed within the application. This allows for the easy viewing of a user’s answers within the application, and allows for the input answers to be searched for based on the id number present within the relevant inputs text file.

XML layout files are used to define the general appearance of different screens within the application. These files define the layout elements that are present on the screen, such as buttons, textViews and imageViews, as well as important values for each element, such as their size, placement on the screen, and default text values [13].

XML is also used for the “android manifests” which is a document within an android document that is used to declare many important aspects of an android application. These declaration include activity declarations, where the activities/screens used in an application must be declared, along with any important values associated with them, and any relevant permissions used or required within the application, such as storage access permissions.

The python programming language is also used to aid in the visualisation and interpretation of the data that is gathered by the application. Using python allows for access to the pandas and matplotlib libraries, which allow for easy implementation of data visualisation and analysis. The pandas library allows for the recorded input data to be placed within a dataframe object, upon which the necessary operations can be performed in order visualise data, as well as to separate data into different dataframes as appropriate [14].

Additionally, the factory method design pattern is used within the project to help in creating the different enemy classes. This design pattern allows for the surfaceview class to interact with the different types of obstacle sub classes using methods that take the generic obstacle class as parameters without having to know which sub class is being used at a particular time. Further information on the factory method design pattern is detailed in the section on system design [15].

Chapter 5 System Specification and Design

The developed application is intended to run on devices running android 12 or higher.

The creation of different types of enemies at random is achieved using the factory method design pattern. The factory method design pattern is a creational design pattern that is used to create instances of a classes sub class, in a situation where the code or method that is creating the class instance may not necessarily know what sub classes need to be created [15].

One of the main advantages of the factory method design pattern is that it allows for the application code to interact solely with the interface or abstract class that is defined as part of the pattern, so that the code will work with any class that extends the abstract class or implements the interface that is used to define the relevant object[15].

As an example, in the developed application, the abstract class “Obstacle” is created. This abstract class cannot be defined directly in the application code, so the only way to access it is by creating the relevant sub classes. To this end, three sub classes were created for the project: a red cloud object, a green cloud object, and a blue cloud object. Each of these objects extend the obstacle class and make use of the methods and variables defined within obstacles, while also defining some of their own operation. The abstract method “interact(Playersprite p)” is listed in the abstract class, and its operations are defined in each of the sub classes. This allows each sub class to interact with the Playersprite object in different ways, using the same method call.

Additionally, an object creation method, which is the Factory Method itself, is also created, which takes in arguments that influence the sub class that will be created by the method. For example, a String object with the text “REDENEMY” will result in the factory method returning a RedEnemy object, which is one of the classes that extend the Obstacle Abstract class. As a result, the factory method pattern allows for object instances to be created without requiring the code to explicitly instantiate a new object directly within the relevant code, rather, the factory method can simply be called when needed, which also aids in the readability of the code.

The structure of the factory method design pattern allows for the base class to be defined as either an abstract class or an interface. An abstract class is a type of class for which instances of that class can only be accessed by creating an instance of a sub class. The required amount of sub classes are also created and extend the base class. These sub classes must then also implement any methods defined in the interface, if one is used, or any abstract methods that are declared in the relevant abstract class.

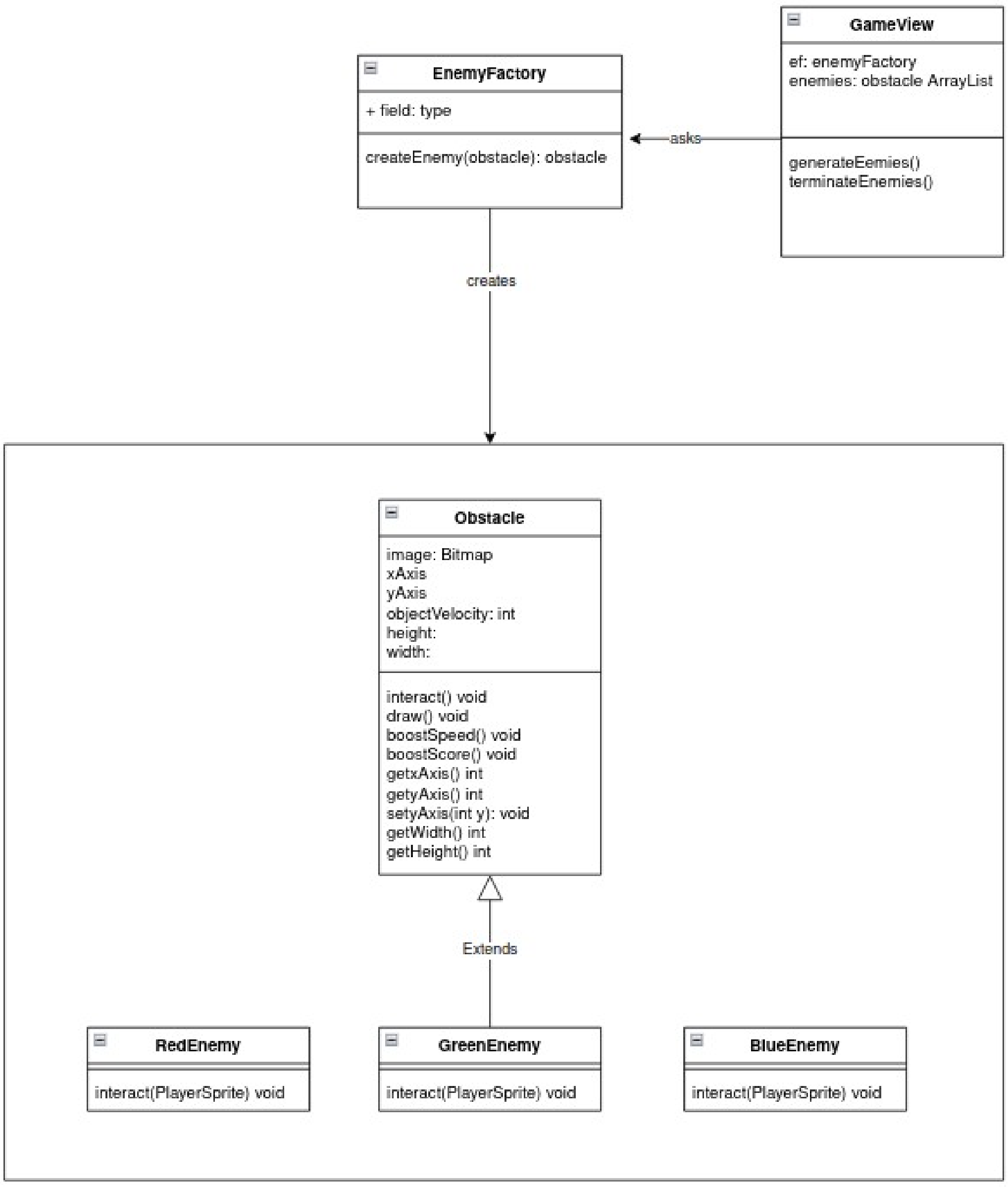


Figure 1. Factory Method Class Diagram

The above class diagram represents the structure of th efactory methid design pattern. It contains the obstacle abstract class, along with three enemy sub classes that extend obstacle. The EnemyFactory class creates enemy objects for use in the GameView class

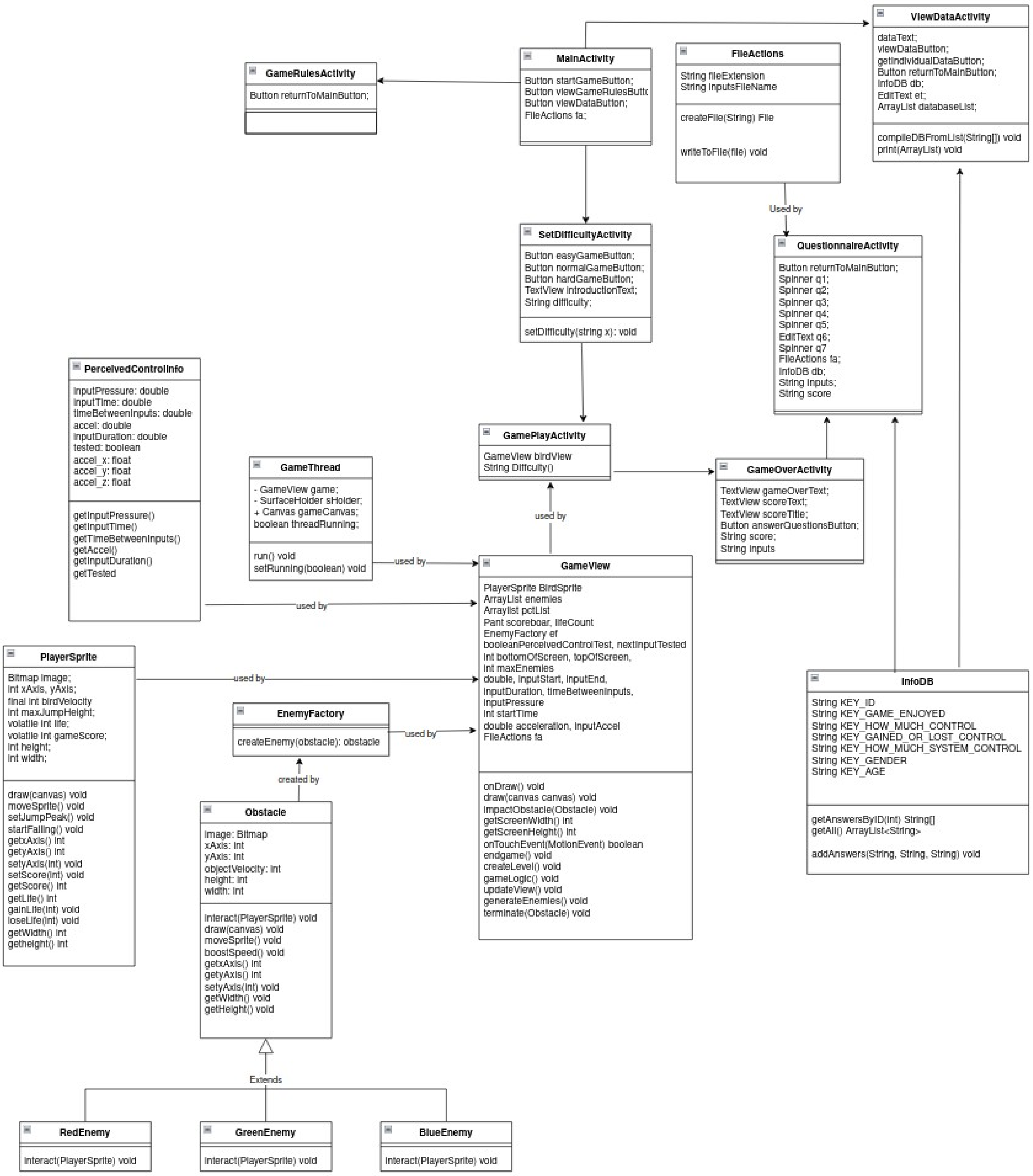


Figure 2. Class diagram representing system design

The following tables describe the key information regarding the classes shown in the class diagram. All of the activity classes make use of the onCreate() method, and all activities that have buttons also use the onClick() method as part of their onClickListeners.

Table 1. Main Activity Class

Class Name	MainActivity

Variables	Button startGameButton;
	Button viewGameRulesButton;
	Button viewDataButton;
	FileActions fa;

Methods	
Relationships to other classes	Used to access set difficulty activity, game rules activity and view data activity
Description	Activity that is booted into upon start of the application, serves as the main menu for the app.

Table 2. set difficulty class

Class Name	SetDifficultyActivity

Variables	Button easyGameButton;
	Button normalGameButton;
	Button hardGameButton;
	TextView introductionText;
	String difficulty;

Methods	setDifficulty(String x)
Relationships to other classes	Accessed by mainActivity, accesses GamePlayActivity.
Description	Activity that is used to set the difficulty level of the game.

Table 3. game play activity class

Class Name	GamePlayActivity
Variables	GameView birdView String Diffculty
Methods	
Relationships to other classes	Instantiates GameView, accessed by setDifficulty activity, and accesses GameOverActivity
Description	Methods associated with gameplay exist within the GameView class.

Table 4. game over activity class

Class Name	GameOverActivity
Variables	TextView gameOverText TextView scoreText TextView scoreTitle Button answerQuestionsButton String score String inputs
Methods	
Relationships to other classes	Accessed from GamePlayActivity by means of the GameView, used to access the Questionnaire activity.
Description	The purpose of this class is to allow a user to view the score that they got in the game before they

	proceed to the Questionnaire.

Table 5. game rules class

Class Name	GameRulesActivity
Variables	Button returnToMainButton;
Methods	
Relationships to other classes	Accessed by MainActivity, and returns to MainActivity
Description	This activity/class is used to describe the basic rules of the game, so that people can understand how the game works before starting to play it.

Table 6. Questionnaire class

Class Name	QuestionnaireActivity

Variables	Button returnToMainButton;
	Spinner q1;
	Spinner q2;
	Spinner q3;
	Spinner q4;
	Spinner q5;
	EditText q6;
	Spinner q7
	FileActions fa;
	InfoDB db;
	String inputs;
	String score;

Methods	compileAnswers()

Relationships to other classes	Accessed by the gameOver activity, returns application to the main activity upon submission of answers
Description	Thsi activity is used to take in the answers to the questions provided in the activity questionnaire, and record them to the applications database for future review.

Table 7. View data class

Class Name	ViewDataActivity

Variables	dataText;
	viewDataButton;
	getIndividualDataButton;
	Button returnToMainButton;
	InfoDB db;
	EditText et;
	ArrayList databaseList;

Methods	compileDBFromList(String[]) void print(ArrayList) void
Relationships to other classes	Accessed by main activity, returns to main activity, creates in-class instance of the appplcication database
Description	This activity allows for th eviewing of questionnaire answer data that is defined and submitted via the QuestionnaireActivity.

Table 8. game view class

Class Name	GameView
Variables	PlayerSprite BirdSprite ArrayList enemies Arraylist pctList Paint scoreboard, lifeCount EnemyFactory ef booleanPerceivedControlTest, nextInputTested int bottomOfScreen, topOfScreen, int maxEnemies double, inputStart, inputEnd, inputDuration, timeBetweenInputs, inputPressure int startTime double acceleration, inputAccel FileActions fa
Methods	onDraw() void draw(canvas canvas) void impactObstacle(Obstacle) void getScreenWidth() int getScreenHeight() int onTouchEvent(MotionEvent) boolean endgame() void createLevel() void gameLogic() void updateView() void generateEnemies() void terminate(Obstacle) void
Relationships to other classes	Instantiated by GamePlayActivity. Instantiates PerceivedControlInfo and GameThread, PlayerSprite and Obstacles
Description	Defines the core gameplay operations associated with the game.

Table 9. game thread

Class Name	GameThread

Variables	- GameView game;
	- SurfaceHolder sHolder;
	+ Canvas gameCanvas;
	boolean threadRunning;

Methods	run() void setRunning(boolean) void
Relationships to other classes	Used/instantiated by GameView
Description	Used to perform gameplay operations without interfering with or blocking necessary operations on the main UI thread

Table 10. obstacle class

Class Name	Obstacle
Variables	image: Bitmap xAxis: int yAxis: int objectVelocity: int height: int width: int
Methods	interact(PlayerSprite) void draw(canvas) void moveSprite() void boostSpeed() void getxAxis() int getyAxis() int setyAxis(int) void getWidth() void getHeight() void

Relationships to other classes	Extended by redEnemy, BlueEnemy, and GreenEnemy.
Description	Base abstract class that is used to define common operations and required abstract methods for relevant sub classes, for the purposes of the factory method design pattern. In the GameView, the obstacle sub classes are typically placed within an ArrayList, where this ArrayList has a varying maximum size based on the selected difficulty: 3 for easy, 5 for normal, and 7 for hard difficulty.

Table 11. enemy factory class

Class Name	EnemyFactory
Variables	
Methods	createEnemy(obstacle): obstacle
Relationships to other classes	Returns object of base type obstacle and extended class of Red Blue or green Enemy
Description	The Factory Method that is the namesake of the factory method design pattern. It is used to create the subclasses of the obstacel class, and returns them in its output, to be saved to a relevant variable as needed. In the GameView, these obstacle sub classes are typically placed within an ArrayList.

Table 12. perceived control info class

Class Name	PerceivedControlInfo

Variables	inputPressure: double inputTime: double timeBetweenInputs: double accel: double inputDuration: double tested: boolean

Methods	getInputPressure()
	getInputTime()
	getTimeBetweenInputs()
	getAccel()
	getInputDuration()
	getTested

Relationships to other classes	Non-determinate number of instances created in GameView
Description	Object that is used to hold input data of potentially different types in one place, all objects of this type typically placed into an ArrayList.

Table 13. answers database class

Class Name	InfoDB
Variables	String KEY_ID String KEY_GAME_ENJOYED String KEY_HOW_MUCH_CONTROL String KEY_GAINED_OR_LOST_CONTROL String KEY_HOW_MUCH_SYSTEM_CONTROL String KEY_GENDER String KEY_AGE
Methods	getAnswersByID(int) String[] getAll() ArrayList<String>

	addAnswers(String, String, String) void
Relationships to other classes	Instantiated by Questionnaire Activity to add entries to database. Instantiated by View Data Activty to View the data entries.
Description	Class that reperesents the local database that is used in the project to record users questionnaire answers. The String key variables listed above represent the columns of the database table, with KEY_ID being an autoincrementing value that is unique for each entry.

Table 14. file actions class

Class Name	FileActions
Variables	String fileExtension String inputsFileName
Methods	createFile(String) File writeToFile(file) void
Relationships to other classes	Used by Questionnaire activity to store input data into files.
Description	Class used to simplify the process of file manipulation. It is used to create files, and write to a file.

Table 15. green enemy class

Class Name	GreenEnemy

Variables	ObjectVelocity: int (defined based on obstacle subType)
Methods	interact(PlayerSprite p) void: overridden from abstract metod in obstacle.
Relationships to other classes	Extends Obstacle. Created and returned within the EnemyFactory.
Description	Obstacle sub class that heals the player chaaracters life upon interacting with it.

Table 16. red enemy class

Class Name	RedEnemy
Variables	ObjectVelocity: int (defined based on obstacle subType)
Methods	interact(PlayerSprite p) void: overridden from abstract metod in obstacle.
Relationships to other classes	Extends Obstacle. Created and returned within the EnemyFactory.
Description	Obstacle sub class that inflicts a large amount of damage to the player chaaracters life upon interacting with it.

Table 17. player sprite character class

Class Name	PlayerSprite

Variables	Bitmap image;
	int xAxis, yAxis;
	final int birdVelocity
	int maxJumpHeight;
	volatile int life;
	volatile int gameScore;
	int height;
	int width;

Methods	draw(canvas) void moveSprite() void setJumpPeak() void startFalling() void getxAxis() int getyAxis() int setyAxis(int) void setScore(int) void getScore() int getLife() int gainLife(int) void loseLife(int) void getWidth() int getheight() int
Relationships to other classes	Instantiated by GameView. Interacts with Obstacle sub-classes.
Description	This class represents the player character within the game.

Table 18. blue enemy class

Class Name	BlueEnemy
Variables	ObjectVelocity: int (defined based on obstacle subType)

Methods	interact(PlayerSprite p) void: overridden from abstract metod in obstacle.
Relationships to other classes	Extends Obstacle. Created and returned within the EnemyFactory.
Description	Obstacle sub class that inflicts a small amount of damage to the player characters life upon interacting with it.

The following sequence diagram describes the general process by which perceived control related data is gathered:

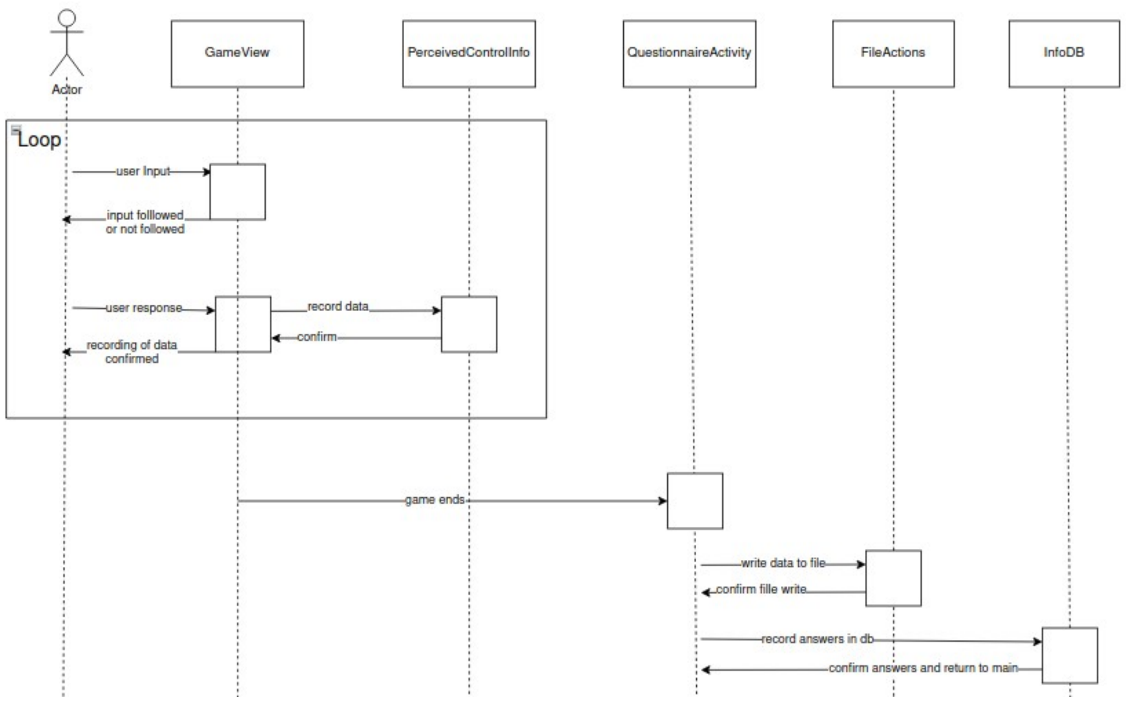


Figure 3. Sequence diagram for input test process

For this diagram, a user provides their input, and and the GameView will either test that input or will not test it. Regardless, the input data is saved to PerceivedControlInfo. This repeats on a loop until the game ends, and the user goes to the questionnaire activity. Once the questions have been answered, the answers are saved to a database, and the input data is saved to a text file using the FileActions class.

The following use case diagram describes the general intended use case of the application:

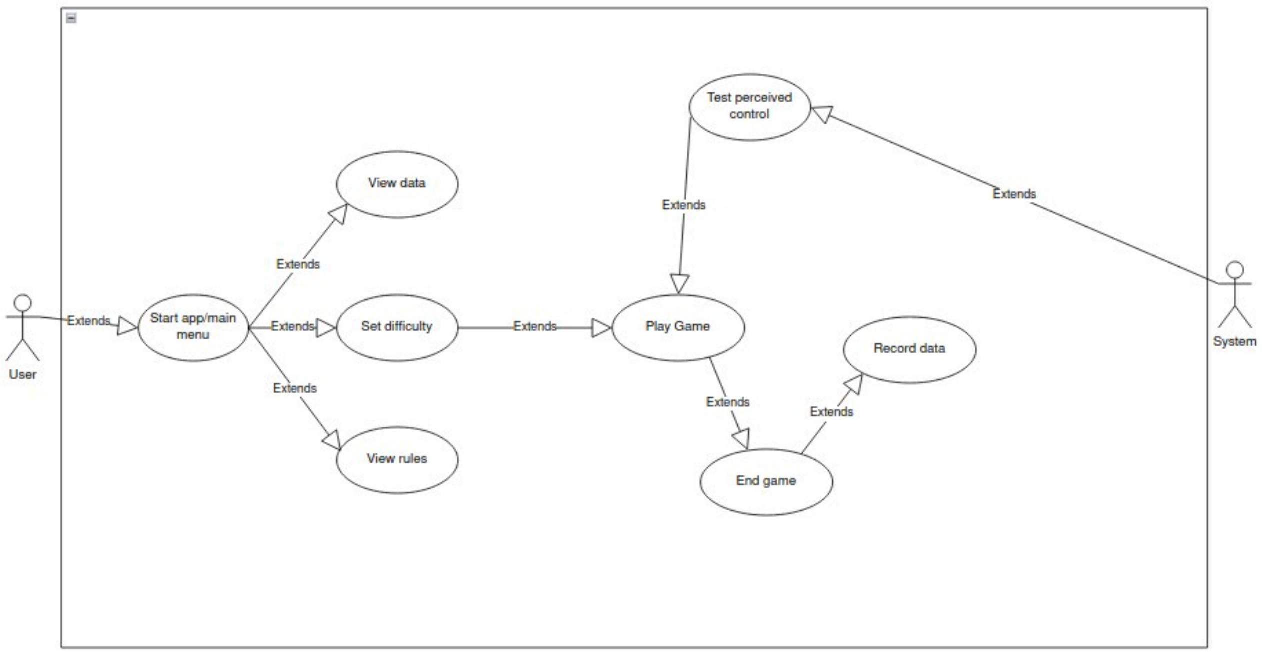


Figure 4. Use case diagram.

In this diagram, the user is the main actor, who starts the use of the app/game. They will be able to view data, view game rules, or start the game. The game starts once the difficulty is selected, and from here, the system will also act to test user inputs accordingly. After the game ends, all of this data is recorded so as to be viewed outside of the game.

Chapter 6 System Implementation

This section will discuss the means by which the features discussed in the technical theory section were implemented within the game, with brief code snippets provided to further discuss this.

Activities were implemented using the following type of declaration with the project:

```
public class MainActivity extends AppCompatActivity {

    Button startGameButton;
    Button viewGameRulesButton;
    Button viewDataButton;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
```

All of the activities extend the AppCompatActivity, and most activities also have layput elements based on the layout elements present in the associated layout XML files, which are declared via setContentView, in the onCreate() method.

Intents were implemented in the project to allow the application to move from one activity to another. Thery were largely implemented as follows:

```
Intent startGameIntent = new Intent(getBaseContext(), SetDifficultyActivity.class);
startActivity(startGameIntent);
```

The intent is declared by passing in a context argument, as well as the class associated with the activity that is meant to be started. The startActivity() method then ends the current activity and starts the new activity by taking the declared activity as its argument.

The putExtras and getExtras methods were implemented as follows to allow variables to be used un more than one activity:

```
Intent gameOverIntent = new Intent(getContext(), GameOverActivity.class);
gameOverIntent.putExtra("score", birdSprite.getScore());
gameOverIntent.putExtra("inputs", fileData);
((Activity) getContext()).finish();
getContext().startActivity(gameOverIntent);
```

In this example, the score and inputs values are obtained via putExtra() and can then be accessed in the next activity as follows:

```
inputs = getIntent().getExtras().get("inputs").toString();

score = getIntent().getExtras().get("score").toString();
```

The use of motionevents to detect when an input was made was implemeted within the GameView class, as part of the OnTouchEvent method, as follows:

```
public boolean onTouchEvent(MotionEvent event)
{
    float x = event.getX();
    float y = event.getY();
```

In addition, the following conditional statements were used to determine whether the inout event was caused by a finger being placed on the screen or being lifted from the screen(i.e, whether an input was starting or ending):

```
if(event.getAction() == MotionEvent.ACTION_DOWN)
{

}

//set up values to be recorded to object list upon the users finger being lifted
else if((event.getAction() == MotionEvent.ACTION_UP) || event.getAction() ==
MotionEvent.ACTION_CANCEL)
{

}
    return true;
}
```

In this case, ACTION_DOWN refers to an input being started and ACTION_UP refers to an input being finished. ACTION_CANCEL refers to an input being ended prematurely, and is used to prevent issues from occuring with inputs ending improperly.

The classes that allowed for the implementation of the playerSprite and obstacles were implemented as follows:

```
public class PlayerSprite {

    private Bitmap image;
    private int xAxis, yAxis;
    private final int birdVelocity = 10;
    private int maxJumpHeight;
    private volatile int life;
    private volatile int gameScore;
    private int height;
    private int width;

    public PlayerSprite(Bitmap bitmap, int x, int y)
    {
        image = bitmap;
        xAxis = x;
        yAxis = y;
        life = 10;
        gameScore = 0;
        height = this.image.getHeight();
        width = this.image.getWidth();
    }
```

The obstacle base class is defined as follows:

```
public abstract class Obstacle{

    public Bitmap image;
    private int xAxis, yAxis;
    public int objectVelocity;
    private int height;
    private int width;

    public Obstacle(Bitmap bitmap, int x, int y)
    {
        image = bitmap;
        xAxis = x;
        yAxis = y;
        height = this.image.getHeight();
        width = this.image.getWidth();
    }

            public abstract void interact(PlayerSprite p);
```

The factory method for creating different sub classes of the obstacle base class was implemented as follows:

The sub classes for the enemy types are first define, with abstact methods properly implemented:

```
public class RedEnemy extends Obstacle {
    public RedEnemy(Bitmap bitmap, int x, int y) {
        super(bitmap, x, y);
        this.objectVelocity = 7;
    }

    @Override
    public void interact(PlayerSprite p)
    {
        p.loseLife(2);
    }
}

public class BlueEnemy extends Obstacle {
    public BlueEnemy(Bitmap bitmap, int x, int y) {
        super(bitmap, x, y);
        this.objectVelocity = 3;
    }

    @Override
    public void interact(PlayerSprite p)
    {
        p.loseLife(1);
    }
}


public class GreenEnemy extends Obstacle {
    public GreenEnemy(Bitmap bitmap, int x, int y) {
        super(bitmap, x, y);
        this.objectVelocity = 5;
    }

    @Override
    public void interact(PlayerSprite p)
    {
        //recover life if life is not at the maximum value
        if(p.getLife() < 10)
        {
            p.gainLife(1);
        }
    }
}
```

The following code snippets represent some examples of how XML was used within the project for the defining of screen layout elements, and necessary declarations within manifests:

```
<LinearLayout
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_centerVertical="true"
    android:orientation="vertical"
    android:layout_alignParentBottom="true">

<Button
    android:id="@+id/startgamebutton"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_marginStart="2dp"
    android:background="@drawable/buttonbg"
    android:text="@string/start_game"
    />
    <TextView
        android:layout_width="match_parent"
        android:layout_height="30dp"
        />
```

This is a sample of the XML from the layout file for the main activity. One of the elements of this layout is a LinearLayout object, which contains a Button and a TextView, for which relevant actions are defined in the Main Actvity class.

The file input and output operations were defined in the FileActions class, to simplify file manipulation within the project, as follows:

```
public class FileActions {
    String inputsFileName;
    //String answersFileName;
    String fileExtension;

    public FileActions()
    {
        inputsFileName = "inputs";
        //answersFileName = "answers";
        fileExtension = ".txt";
    }

    //create file object
    public File createFile(String fileName)
    {
        File dir = 
Environment.getExternalStoragePublicDirectory(Environment.DIRECTORY_DOCUMENTS);
        File file = new File(dir, fileName);
        return file;
    }



    //write to file as necessary
    public void writeToFile(File file, String s)
    {
        try
        {
            FileWriter myWriter = new FileWriter(file);
            myWriter.write(s);
            myWriter.close();
        }
        catch(Exception exception)
        {
            exception.printStackTrace();
        }
    }
```

Threading was implemented in order to prevent any performane issues resulting forom the long term continuous game operations:

```
public class GameThread extends Thread{

    private GameView game;
    private SurfaceHolder sHolder;
    public static Canvas gameCanvas;
    boolean threadRunning;
public GameThread(GameView v, SurfaceHolder s)
{
    super();
    this.game = v;
    this.sHolder = s;
}



//method to start execution of thread operations
@Override
public void run() {

    while (threadRunning) {
        //check if surfaceHolder is valid before doing any operations
        if(sHolder.getSurface().isValid())
        {
            gameCanvas = sHolder.lockCanvas();
            game.updateView();
            game.draw(gameCanvas);
            sHolder.unlockCanvasAndPost(gameCanvas);
        }

    }
}

//set running status of thread in order to start or stop it
    public void setRunning (boolean running)
    {
        this.threadRunning = running;
    }



}
```

The accelerometer sensor was implemented in the android code as follows, in order to allow for the acceleration data to be gathered an interpreted based on relevant calculations:

```
private void startAccelerometerSensor(SensorEvent event)
{
    float values[] = event.values;

    accel_x = values[0];
    accel_y = values[1];
    accel_z = values[2];

    acceleration = Math.sqrt((accel_x * accel_x) + (accel_y * accel_y) + (accel_z *
accel_z));
}
```

Perceived control information was gathered as part of the onTouchEvent method as follows:

The following statements defined the data to be gathered:

```
inputAccel = acceleration;

//get input pressure
inputPressure = event.getPressure();

//get start time for input relative to game start time
inputStart = (((int)System.currentTimeMillis())/1000.0) - startTime;

//value for inputend at this point is the end time for the previous input
timeBetweenInputs = inputStart - inputend;

inputend = (((int)System.currentTimeMillis())/1000.0) - startTime;
inputduration = (inputend - inputStart);
boolean thisInputTested = nextInputTested;
```

Meanwhile the following object declaration collects the data in one object, which is then added to an ArrayList:

```
PerceivedControlInfo pct = new PerceivedControlInfo(thisInputTested, inputPressure,
inputduration, timeBetweenInputs, inputStart, inputAccel, accel_x, accel_y, accel_z);
pctList.add(pct);
```

The android database was implemented to hold the answers provided by the user after completion of the game. Each column is initially defined by a number of KEY string variables:

```
public class InfoDB {
    public static final String KEY_ID = "id";
    public static final String KEY_GAME_ENJOYED = "game_enjoyed";
    public static final String KEY_HOW_MUCH_CONTROL = "how_much_control";
    public static final String KEY_GAINED_OR_LOST_CONTROL = "gained_or_lost_control";
    public static final String KEY_HOW_MUCH_SYSTEM_CONTROL =
"how_much_control_for_system";
    public static final String KEY_GENDER = "gender";
    public static final String KEY_AGE = "age";
    public static final String KEY_ABLE_TO_REGAIN_CONTROL = "able_to_regain_control";
    public static final String KEY_SCORE = "score";

    private Context context;
    private ModuleDBOpenHelper moduleDBOpenHelper;

    public InfoDB(Context context){
        this.context = context;
        moduleDBOpenHelper = new ModuleDBOpenHelper(context,
ModuleDBOpenHelper.DATABASE_NAME, null,
ModuleDBOpenHelper.DATABASE_VERSION);
    }
```

```
public void addAnswers(String ans1, String ans2, String ans3, String ans4, String ans5,
String ans6, String ans7, String score)
```

```
{
    ContentValues newAnswers = new ContentValues();
    newAnswers.put(KEY_GAME_ENJOYED, ans1);
    newAnswers.put(KEY_HOW_MUCH_CONTROL, ans2);
    newAnswers.put(KEY_GAINED_OR_LOST_CONTROL, ans3);
    newAnswers.put(KEY_HOW_MUCH_SYSTEM_CONTROL, ans4);
    newAnswers.put(KEY_GENDER, ans5);
    newAnswers.put(KEY_AGE, ans6);
    newAnswers.put(KEY_ABLE_TO_REGAIN_CONTROL, ans7);
    newAnswers.put(KEY_SCORE, score);

    SQLiteDatabase db = moduleDBOpenHelper.getWritableDatabase();
    db.insert(ModuleDBOpenHelper.DATABASE_TABLE, null, newAnswers);
}
```

The following methods also allow for values to be retrieved from the database:

```
public ArrayList<String> getAll() {


public String[] getAnswersByID(Integer id)
{
```

Finally, and SQLiteOpenHelper objects is used to help create the database as follows:

```
private static class ModuleDBOpenHelper extends SQLiteOpenHelper {

    private static final String DATABASE_NAME = "inputDatabase.db";
    private static final String DATABASE_TABLE = "inputs";
    private static final int DATABASE_VERSION = 1;
    private static final String DATABASE_CREATE = "create table " +
            DATABASE_TABLE + " (" + KEY_ID +
            " integer primary key autoincrement, " +
            KEY_GAME_ENJOYED + " string not null, " +
            KEY_HOW_MUCH_CONTROL + " string not null, " +
            KEY_AGE + " string not null, " +
            KEY_HOW_MUCH_SYSTEM_CONTROL + " string not null, " +
            KEY_GENDER + " string not null, " +
            KEY_ABLE_TO_REGAIN_CONTROL + " string not null, " +
            KEY_GAINED_OR_LOST_CONTROL + " string not null) ";
```

This object helps to define the columns and types of data that are used in the database table, with the ID key being the primary key for the table, and also being declared as an autincrementing value, meaning that it will be defined automatically for each row of the database, in increasing number for each concurrent entry in the database.

Python was used in the developed python notebook to read the inputs data generated from the app into a dataframe object. This dataframe object could then be plotted, operated on, and split up as needed. The follwoing code snippet shows the dataframe being defined, and then being split into tested and untested inputs.

```
data = pd.read_csv('inputs1.txt', sep = ", ", header = 0)
inputs_tested = data[data["PCT"]==True]
inputs_untested = data[data["PCT"]==False]
colors={True:'red', False: 'blue'}
print(data)
```

As an example, input durations are then plotted and operated on as follows, with mean values and standard deviations being calculated as follows:

```
inputs_tested['Input Duration'].plot()
plt.show()
print("Mean of tested duration: ", inputs_tested['Input Duration'].mean())
print("Standard deviation of tested duration: ", inputs_tested['Input
Duration'].std())

inputs_untested['input duration'].plot()
```

```
plt.show()

print("Mean of untested duration: ", inputs_untested['Input
Duration'].mean())

print("Standard deviation of untested duration: ", inputs_untested['Input
Duration'].std())
```

Thes same operations are also applied to the time between inputs, the input durations, and the acceleration values derived from the accelerometer sensor.

Finally a perceptron algorithm is declared and used within the python code to determine if a machine learning based algorithm can be used to accurately determine if a given row of raw input data is a tested or untested input, as follows:

```
X = data.iloc[:,2:5].values

y = data.iloc[:,0].values

scaler = StandardScaler()

X = scaler.fit_transform(X)

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33,
random_state=42)

ppn = Perceptron()

ppn.fit(X_train, y_train)

X_pred = ppn.predict(X_test)

print("Accuracy score: ", accuracy_score(y_test, X_pred))
```

Standard scaling was also applied to the data before being used with the perceptron, as it was found that this provided a small increase in accuracy for the algorithm

Chapter 7 Project Results

This section will discuss the final results of the project work thus far, and detail a simple walkthrough of the general runtime process of the applications use.

As a final result of the project work, a rudimentary game has been developed that allows a user to tap the screen to make a bird character fly upwards in order to avoid abstacles. The game, at random, does not follow a users input(i.e the bird will randomly not jump), with all of the relevant input data being recorded, including details on whether or not the current input is a response to a prior input not being followed, for later recording in an external text file.

An additional result of the project is the development of a python notebook that is used to analyse the input data, as well as to interpret and draw the relevant conclusions from this data. This notebook can be viewed using the releavnt application, such as Jupyter notebook or Google colaboratory, to view the outputs of the python code without need to be able to run the code directly.

The overall runtime of the application can be described as follows:

Upon starting the application, the first activity to be started is the main activity. Frome here, the user can click the relevant buttons to start the game, view the game rules, or view the questionnaire.

When the option to start the game is chosen, the main activity will first clear the inputs text file, as this text file is intended to only store data regarding one subject at a time, and selecting the option to start the game from the main activity is intended to start a new session for a new individual.

After the file information is cleared, the user will be prompted to select a dificulty level for the game, from easy, normal and hard difficulties. These difficulty levels influence the amount of enemies that appear on the screen at any given time, 3 enemies for easy difficulty, 5 enemies for normal, and 7 enemies for hard difficulty.

After the difficulty level is selected, the game starts, and the user will tap the screen in order to allow the bird to avoid the different obstacles. As previously described, inputs will randomly not be followed, and the inputs will be recorded to determine how the user reacts to their inputs not being followed by the game.

When the player characters life hits zero, the game ends, and the player is taken to the Game over screen. From here, the user can view the score that they achieved in the game or proceed to the Questionnaire.

Once the user has determined that they are finished playing the game, they can proceed to the questionnaire, where they are required to answer a small set of questions regarding the level of control that they believed that they had over the game. Once the user has submitted their answers for the questionnaire, the answers are recorded in the application database, and the application return to the main menu.

From this point, a user can then enter the activity that is used to view data, where their answers can be viewed in the application directly. This data can be retrieved based on the user id number (in the event that answers relative to a certain set of inputs need to be viewed) or by name(in the event that user wishes to review the answers that they have given).

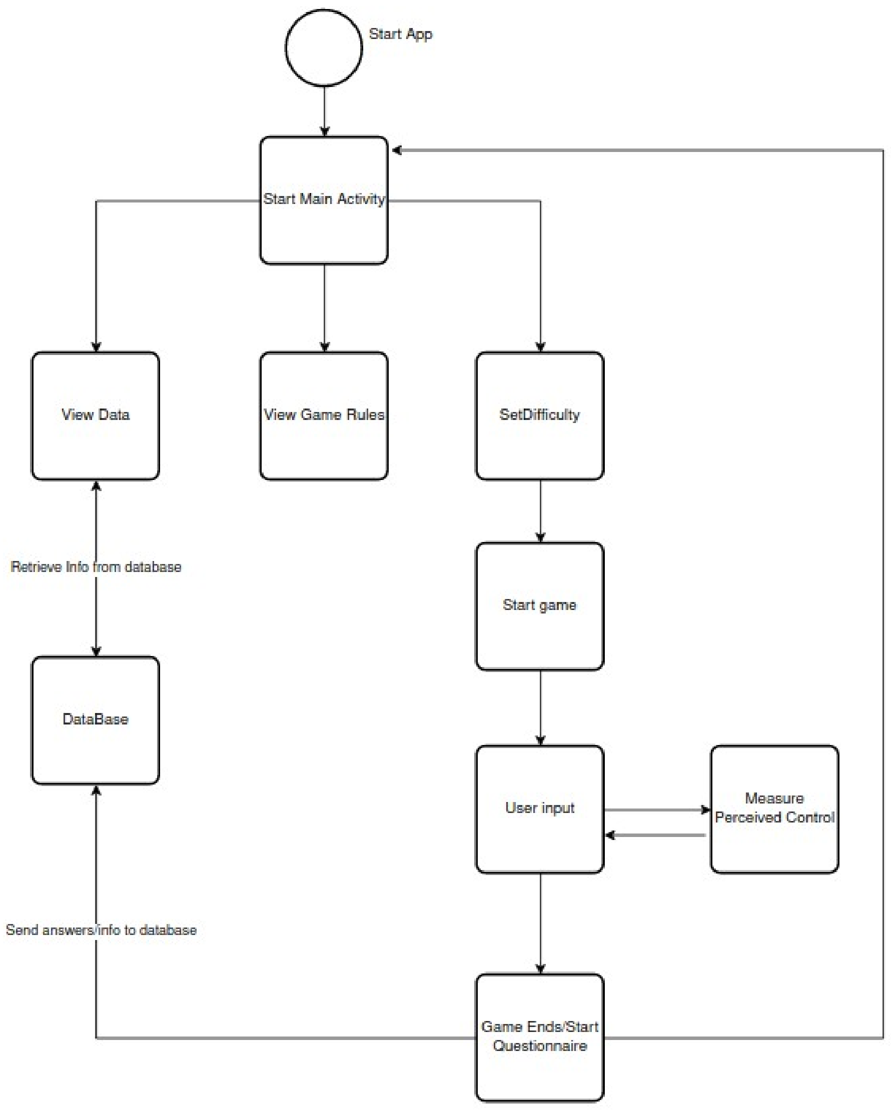

Start App
Start Main Activity
View Data
View Game Rules
SetDifficulty
Retrieve Info from database
Start game
DataBase
User input
Measure Perceived Control
Send answers/info to database
Game Ends/Start Questionnaire

Figure 5. flowchart of application process

Chapter 8 Test Data Visualisation/Analysis

This chapter will discuss the input results generated by the personal testing of the application, and will discuss a potential method of visualising this data, along with graphs to display the potential appearance of this data.

Given that other people did not participate in the testing of the application, it is difficult to ascertain the validity of the gathered test data in the context of measuring perceived control, as personal awareness of the system functionality will likely affect the gathered input data. The following data that will be discussed in this section is, as such, intended to demonstrate a proof of concept for the gathering and visualisation of input data, and how such data might be expected to appear.

The python notebook that was developed is used to compare each portion of the input data to the time of the input, for example, input duration, input pressure, or time between inputs on the y axis, being plotted against input time on the x axis, to represent the values of input duration over time.

Outside of measuring input values over time, the python notebook is also used to compare each different aspect of the perceived control data against each other, such as input pressure being compared against input duration, to determine if there is any correlation or link between the values of different inputs.

Additionally, a scatterplot can be used for such visualisation of data, as with a scatterplot, each datapoint can be colour coded such so that tested inputs(inputs that are a response to a previous input not being followed by the game) are coloured red, and untested inputs or coloured blue.

This can allow for data points to be generally compared and contrasted with each other, so that information about inputs, and a particular subjects input habits, can be inferred from visually separable input values.

Such a scatter plot would be defined in the python code as follows:

```
data.plot(kind = 'scatter', x = 'input time', y = 'Input Duration', c = data.PCT.map(colors))
```

The resulting plot would then look something like this:

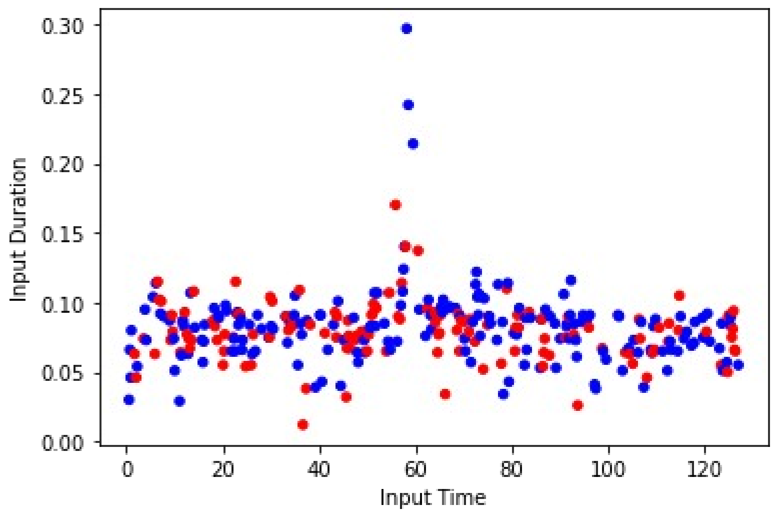


Figure 6. Scatterplot of duration vs. time(Seconds).

This graph, while likely insufficient by itself, would aid an overall analysis of the input data by allowing for the visualisation of potential trends in the data.

In terms of directly comparing tested input data with untested input data, the pandas dataframe that is used in the python notebook can be further split into two separate dataframes, based on whether each set of inputs represents a tested input or not. Following this, the standard deviations and means can be calculated for each set of inputs, both tested and untested, to determine if there is any major differences between th different datasets.

For example, the standard deviations and mean for the input duration of tested inputs would be calculated as follows:

```
print("Mean of tested duration: ", inputs_tested['Input Duration'].mean())

print("Standard deviation of tested duration: ", inputs_tested['Input Duration'].std())
```

The example result from this would be as follows:

```
Mean of tested duration:  0.07955303030691785
Standard deviation of tested duration:  0.02075274815451 2345
```

Additionally, this data can also be plotted on a line graph, which is displayed as follows:

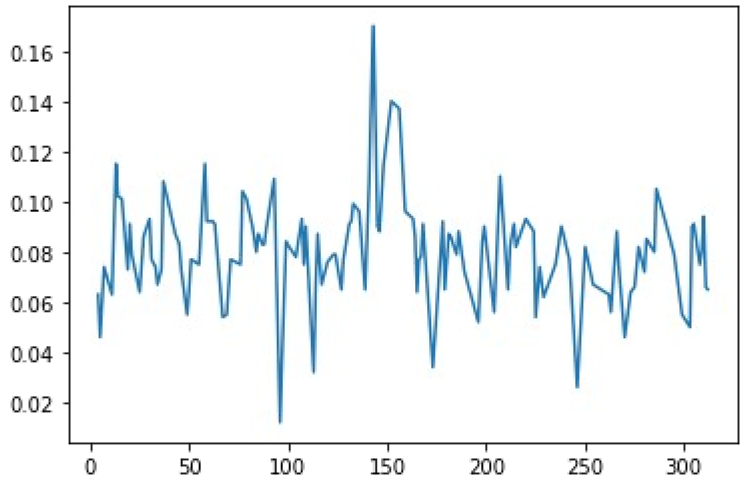


Figure 7. Tested input duration(in seconds)

These processes can also be applied to the untested data to produce the following outputs:

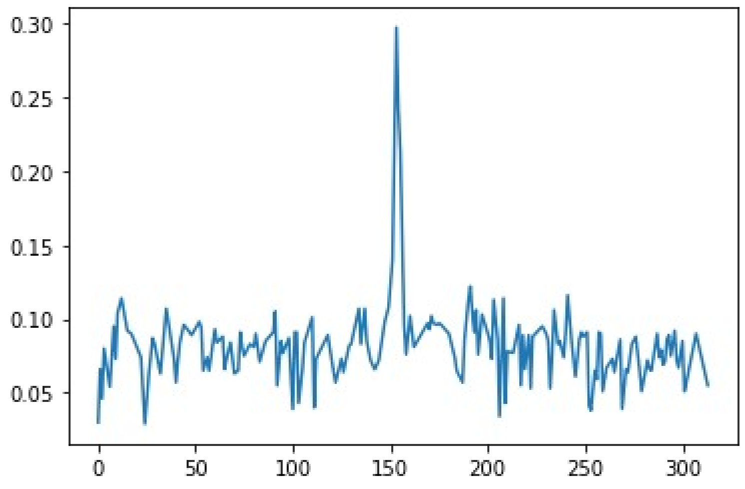


Figure 8. untested input duration(in seconds)

```
Mean of untested duration:  0.08236813187473811
Standard deviation of untested duration:  0.029269969826023454
```

Based on this sample data generated by personal testing of the application, it appears that the mean of tested input duration is slightly lower than the mean for untested input duration, though the values appear to be very close to each other. This is also the case for the standard deviation for each of these datasets. This suggests that, for someone who has an intricate understanding of how the game works, there would not be a significant difference in input duration for tested and untested inputs.

These processes can also be applied to device acceleration, input pressure and time between inputs.

The calculation of device acceleration on personally created testing data provides the following outputs:

For tested inputs:

```
Mean of tested acceleration:  9.782427279865217
Standard deviation of tested acceleration:  0.5201148299366424
```

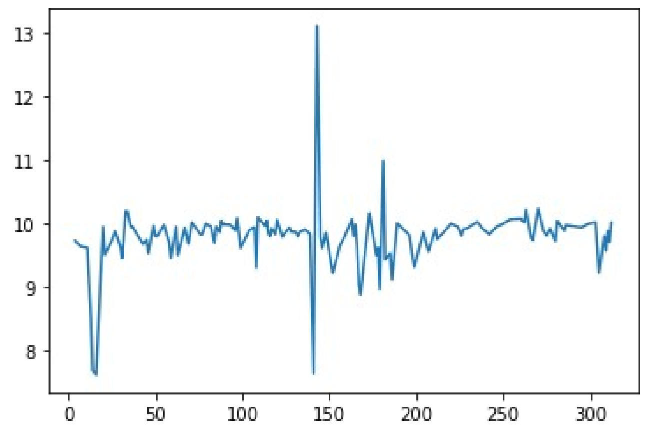


Figure 9. Tested acceleration(m/s^2)

And for untested acceleration:

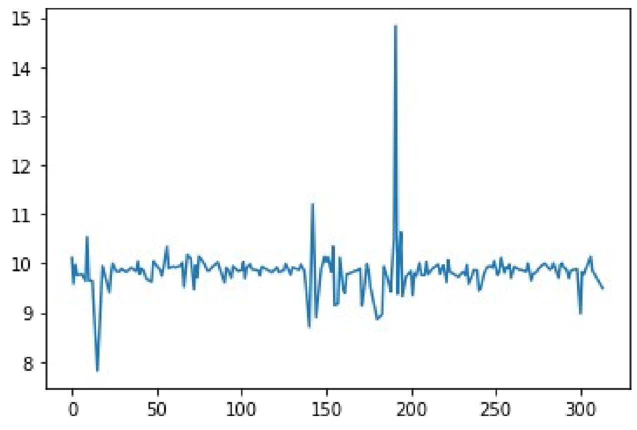


Figure 10. Untested acceleration(m/s^2)

```
Mean of untested acceleration:  9.837555122975882
Standard deviation of untested acceleration:  0.49491938317761547
```

The mean of tested acceleration is slightly lower here than for untested acceleration, with standard deviation being slightly higher for tested inputs

The calculation of input pressure on personally created testing data provides the following outputs:

For tested data:

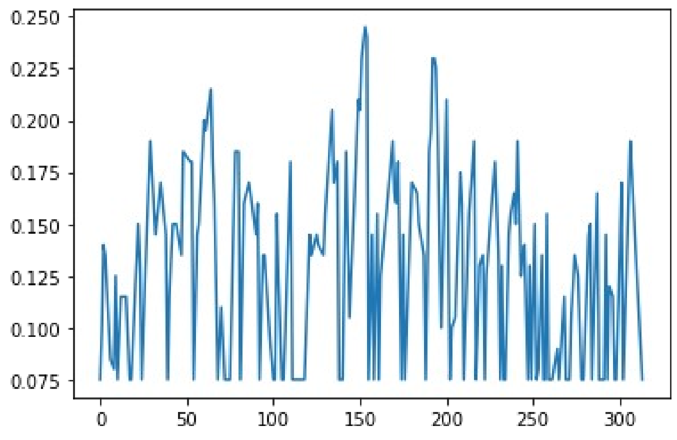


Figure 11. untested pressure

```
Mean of untested pressure:  0.1283791163465479
Standard deviation of untested pressure:  0.046421491722884727
```

For untested data:

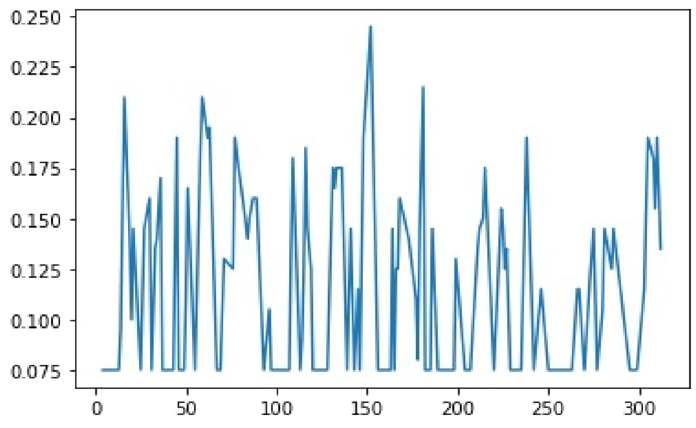


Figure 12. Tested pressure

```
Mean of tested pressure:  0.12041666268399268
Standard deviation of tested pressure:  0.0441200724816754
```

The mean for untested pressure appears to be marginally higher than for tested input pressure, which also appears to be the case for the standard deviation, though the values are still very similar.

The calculation of time between inputs on personally created testing data provides the following outputs:

For tested time between inputs:

fig 13. Tested time between

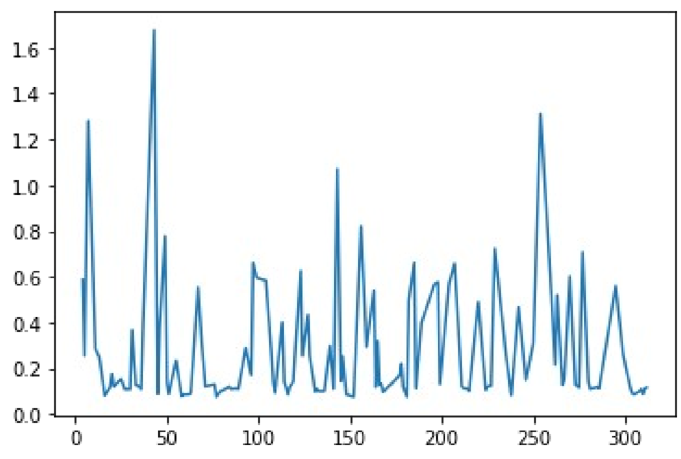


Figure 13. Tested time between inputs(seconds)

```
Mean:  0.2719015151466892
Standard Deviation:  0.26962283126947784
```

For untested time between inputs:

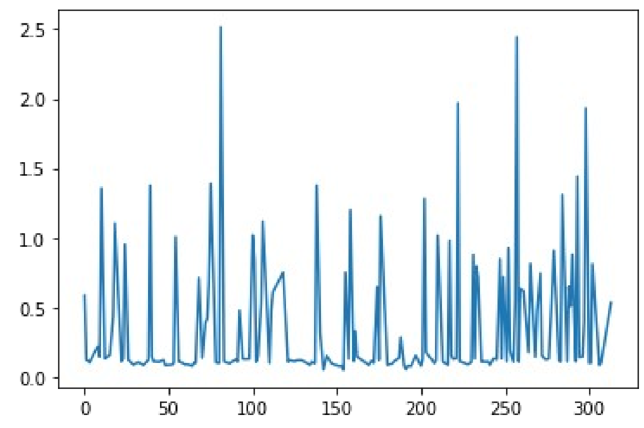


Figure 14. Untested time between inputs(seconds)

Mean: 0.3610109890053806

```
Standard Deviation:  0.45252601023176653
```

The mean of the time between inputs for the tested inputs appears to be smaller than the same value for untested inputs, with standard deviation for tested inputs being roughly 0.2 lower than that for untested inputs(which lay at about 0.45 in comparison).

Overall, there does not appear to be a major difference in metrics for tested input data and untested input data for the input data that was generated by personal testing. The best guess as to why this is the case is that the personal knowledge of how the system works is influencing the data that is generated by personal testing, thereby creating a potential bias in the input data, compared to what might be seen from a user who is not aware of the fact that the game will not follow certain input sat random.

Additionally, a perceptron algorithm was used in the python notebook to determine if a machine learning algorithm could accurately determine if a given input was a tested or untested input. The perceptron algorithm, after being given data that had been put through a StandardScaler, seemed to indicate an approximately 57.69% accuracy on determining if a given input was tested or untested, based on the following output:

```
Accuracy score:  0.5769230769230769
```

This score suggests that the perceptron may not be able to reliably predict whether an input is tested or untested, as a result that is this close to 50% would essentially amount to a “coin toss” guess of sorts.

Chapter 9 Visual/Art Assets used/created for the project

A number of different artistic/visual assets were used in this application to visualise several different aspects of the applications visual design, such as the button shapes, game logo, background, player character design, and obstacle design. These assets were created using piskel, which is an online pixelated graphics creation tool.

The below figure, for example, is representative of the bird player character within the game, with the size of the image appropriately scaled down to best suit the size of the device screen:

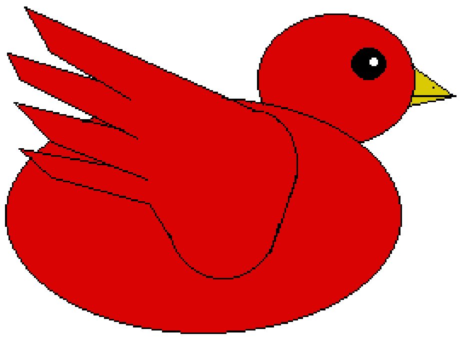


Figure 15. player character design

Additionally, the following design is used to create a customised shape for the buttons that are created within the application:

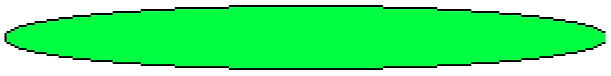


Figure 16. Button Design

The colour of the background of each application activity is also defined by using a plain single coloured canvas created with in piskel:

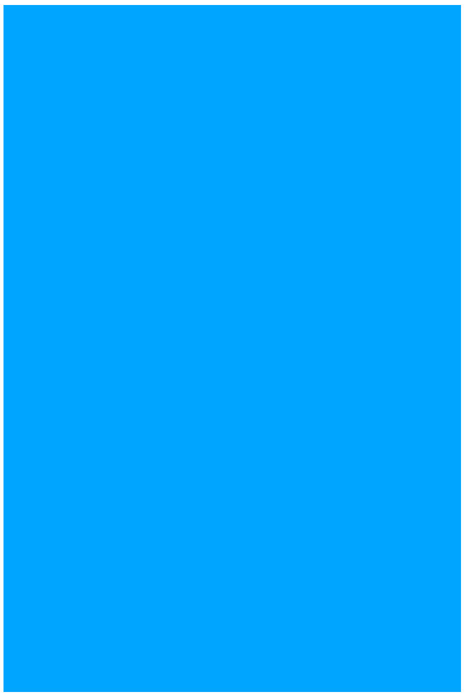


Figure 17. Screen Background Design

The following design is used to aid in the creation and visual representation of the green obstacle objects within the core gameplay. As with the bird character sprite, the size of this image is scaled to down to a suitable size relative to the screen upon which the game is being played.

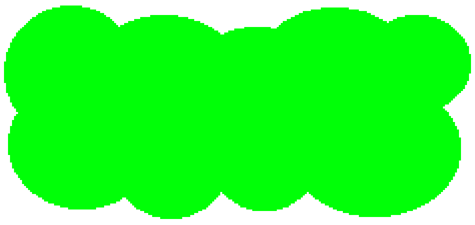


Figure 18. Green Obstacle design

The following design is used to aid in the creation and visual representation of the red obstacle objects within the core gameplay. As with the bird character sprite, the size of this image is scaled to down to a suitable size relative to the screen upon which the game is being played.

Figure 19. Red Obstacle design

The following design is used to aid in the creation and visual representation of the red obstacle objects within the core gameplay. As with the bird character sprite, the size of this image is scaled to down to a suitable size relative to the screen upon which the game is being played.

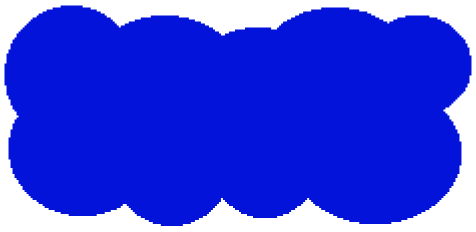


Figure 20. Blue Obstacle design

Finally, the following image is used to create a log for the game, to help attach an identity or name to the game itself.

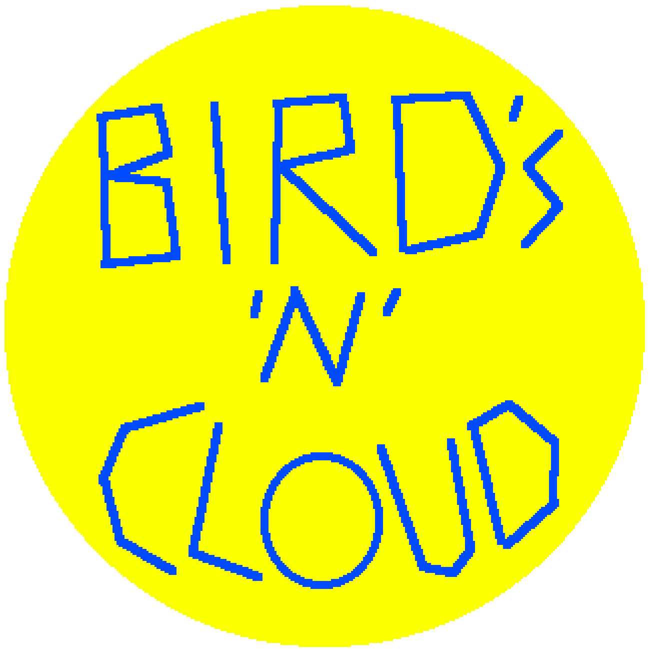


Figure 21. Logo Design

Chapter 10 Conclusions and future work

Overall, I am quite satisfied with what I have accomplished with this project. I feel that my programming skills have improved significantly over the course of the project, and I feel that I have gained a deeper insight into some of the core concepts behind android development, and, more specifically game development in android. I also found the concepts and information around perceived control to be very interesting. I also appreciated the opportunity to work on a long form project such as this over the course of the academic year.

I also feel that my general work ethic, especially in regards to my consistency in maintaining a good level of momentum over the course of a project, has improved as well.

In terms of future work, one feature that I would have liked to implement in the game, if more time were available, would be a ‘bad luck protection’ system. When the game randomly does not follow a user’s inputs, there is no guarantee that the next input will be followed, nor is there a guarantee that a user will not experience a series of inputs that are not followed.

The bad luck protection feature would be intended to ensure that a user does not experience a series of unfollowed inputs, by maintaining a maximum count of inputs that will not be followed in direct sequence.

Another feature that I would potentially like to look into if the project were to continue on for a longer period would be a remotely accessible database. The database that was used within the project was a local “application internal” database that is only used on the specific android device upon which the application runs. This would make the process of transferring data to alternative devices for interpretation and visualisation much easier, as well as potentially offer the opportunity to look into the secure transfer of data across the internet, as would be relevant to my course of study.

Additionally, I would also look into expanding the perceived control questions, and changing the format of the answers to a numbered scale, instead of having answers based in simple phrases, as such numbered answers, which would likely be on a scale of either 1 to 10 or 1 to 100, would then be able to be operated on in order to determine an average scale for each answer.

I would also look into alternative methods of measuring input pressure in the application, or determining another type of data that would serve as an equivalent or better metric then input pressure. The reason for this is that, in android, measuring input pressure seems to be hardware dependant, where some devices return a binary value stating whether pressure was applied, while some will return a variable value as required.

Additionally, while the automatic use of file input and output was effective in creating uniquely identifiable text files, it might potentially be better to include these input values as one string in a column of the application database. This would potentially allow not only for the input text files to be created as needed, but for all input data from every user to be combined together to analyse any potential trends in data. This would be an aspect of input data persistence that I would be keen to look into if I were to either have more time for the project, or if I were to conduct this project again.

References

[1] F. Pagnini, et.al, “Perceived Control and Mindfulness: Implications for

Clinical Practice”, 2016, available:
https://www.apa.org/pubs/journals/features/int-int0000035.pdf

[2] wikipedia.org, “Perceived Control”, last edited: 7 Aug 2022, available:

https://en.wikipedia.org/wiki/Perceived_control

[3] P. Toprac, “The Psychology of Control and Video Games”, Jan. 2013,

available:
https://www.researchgate.net/publication/288841120_The_Psychology_of_

Control_and_Video_Games

[4] B. A. Davis, Development and Validation of a Scale of Perceived

Control Across Multiple Domains, 2004, available:

https://digitalcommons.pcom.edu/cgi/viewcontent.cgi?
article=1036&context=psychology_dissertations

[5]Sinicki A. (2017), androidauthority.com, “How to write your first Android game in Java”, available:https://www.androidauthority.com/android-game-java-785331/

[6]Sinicki A. (2018), androidauthority.com, “Let's build a simple Flappy Bird clone in Android Studio”, Available at: https://www.androidauthority.com/flappy-bird-clone-android-studio-830585/

[7]salinda93, github.com, “The Flying Fish Game”, Available at:
https://github.com/salinda93/The-Flying-Fish-Game

[8] developer.android.com, last updated: 2022, “Introduction to activities”, available: https://developer.android.com/guide/components/activities/intro-activities

[9]developer.android.com, last updated: 2023, “Intents and Intent Filters”, available: https://developer.android.com/guide/components/intents-filters

[10]developer.android.com, last updated: 2023, “MotionEvent”, available: https://developer.android.com/reference/android/view/MotionEvent

[11]developer.android.com, last updated: 2023, “SurfaceView”, available: https://developer.android.com/reference/android/view/SurfaceView

[12]developer.android.com, last updated: 2023, “Motion sensors”, available: https://developer.android.com/guide/topics/sensors/sensors_motion

[13] developer.android.com, last updated: 2023, “Layouts ”, available: https://developer.android.com/develop/ui/views/layout/declaring-layout

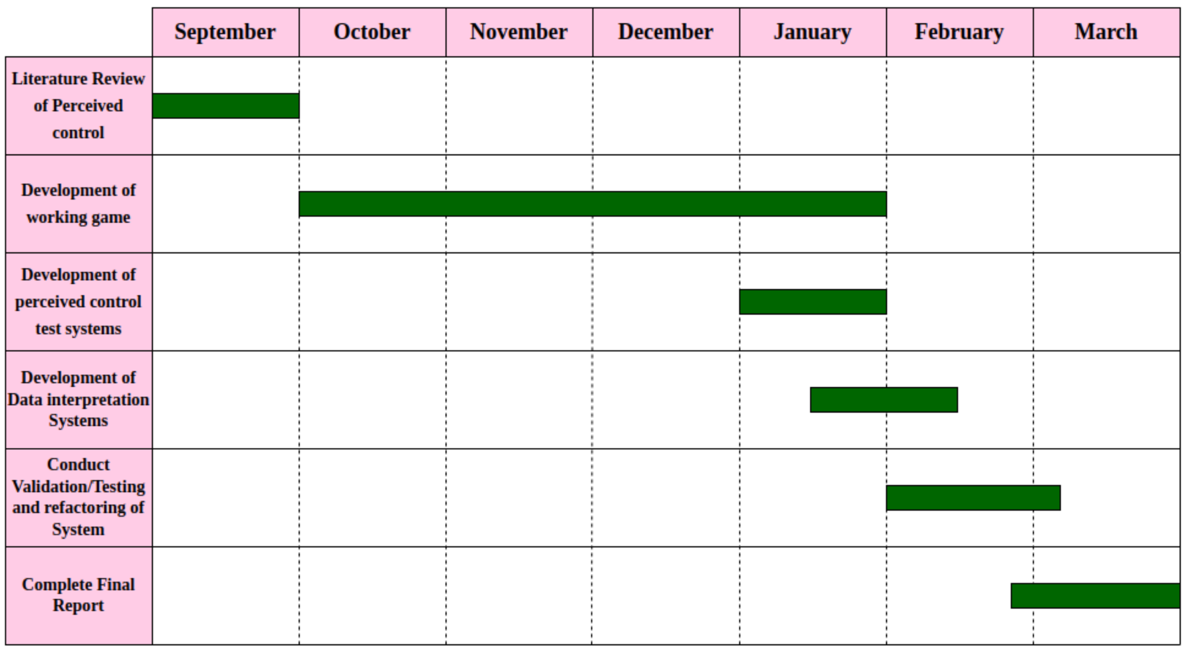
[14]pandas.pydata.org, n.d, “pandas.DataFrame”, available: https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.html

[15]javatpoint.com, n.d, “Factory Method Pattern”, available: https://www.javatpoint.com/factory-method-design-pattern

Appendices

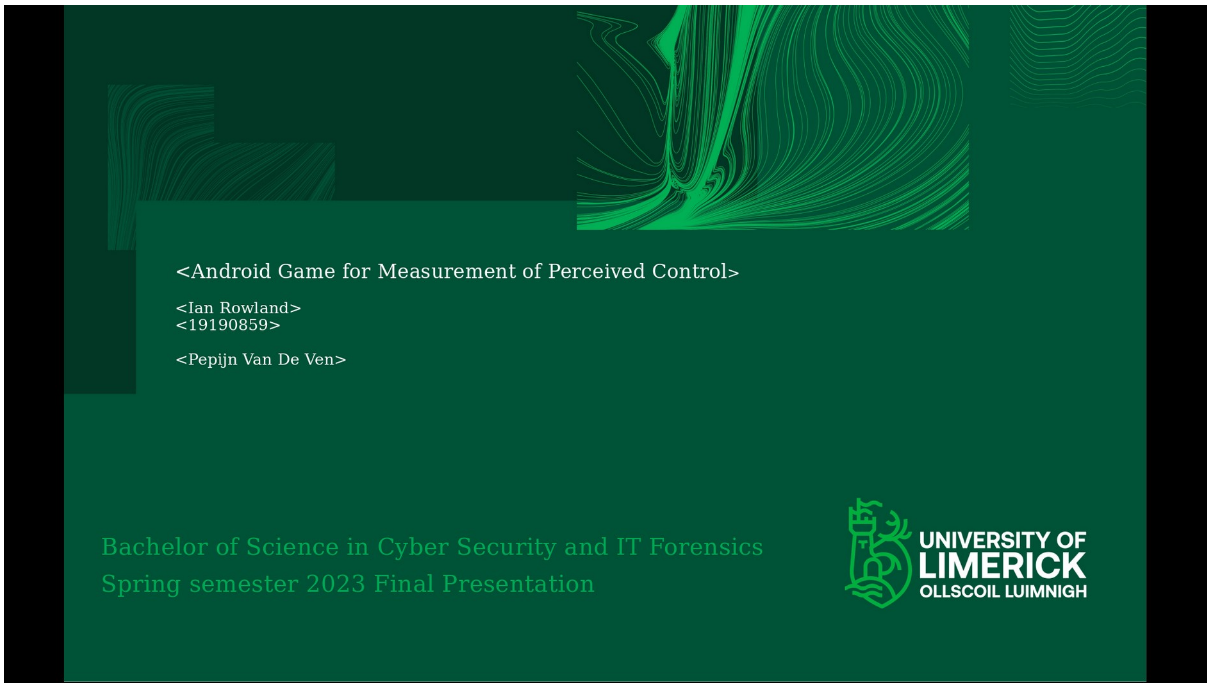
Appendix A: Updated project Gantt chart

Replace this text with the project Gantt chart. This will be an update of the Gantt chart presented in the interim report, and reflect changes made during the project implementation. Appendix to start on odd number page.



Appendices 2

Appendix B: Final presentation slides



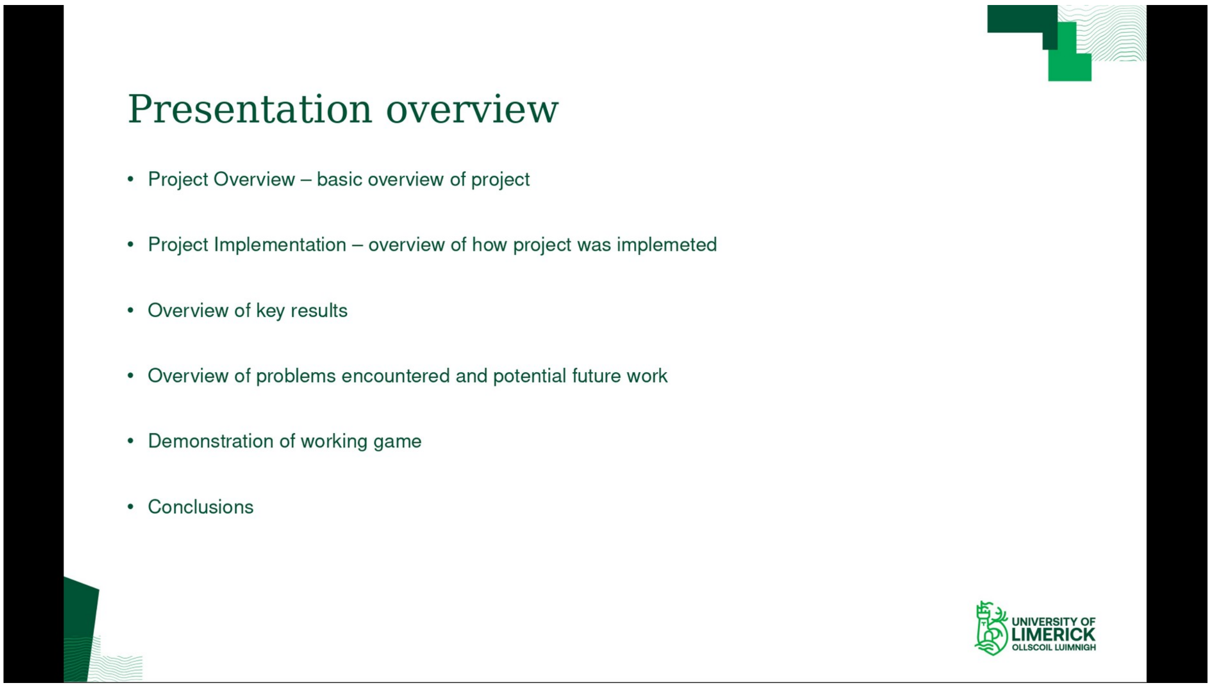
<Android Game for Measurement of Perceived Control>

<Ian Rowland>
<19190859>

<Pepijn Van De Ven>

Bachelor of Science in Cyber Security and IT Forensics
Spring semester 2023 Final Presentation

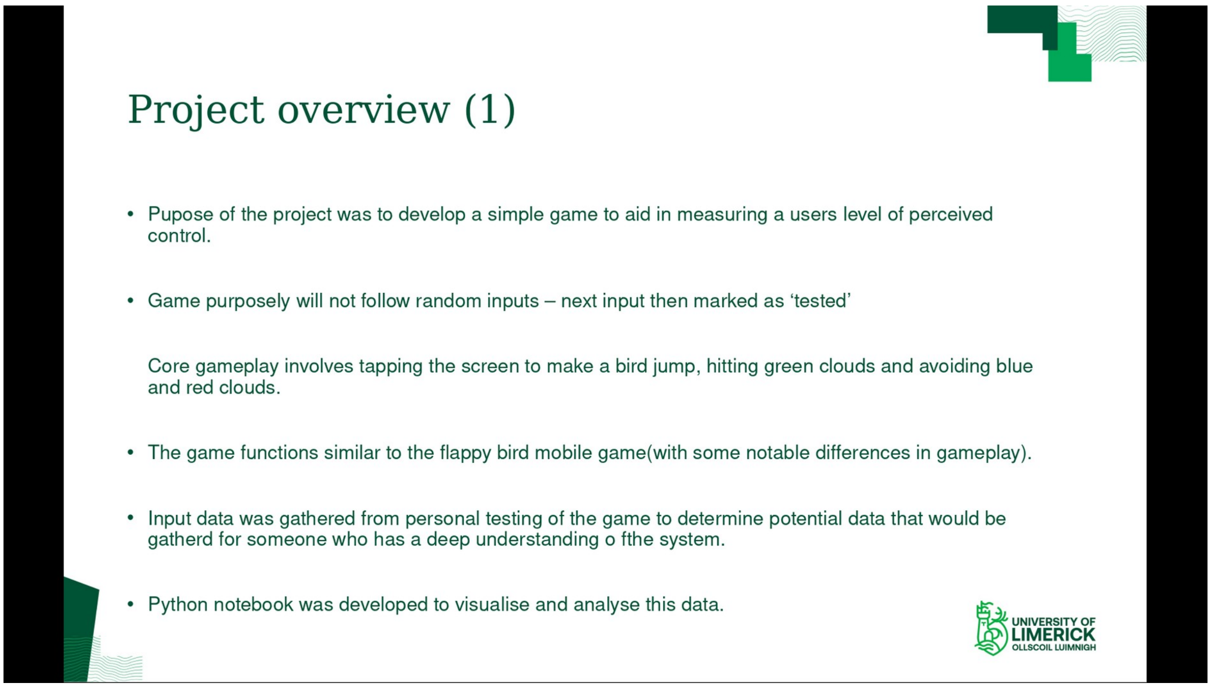
UNIVERSITY OF LIMERICK
OLLSCOIL LUIMNIGH



Presentation overview

- Project Overview – basic overview of project
- Project Implementation – overview of how project was implemeted
- Overview of key results
- Overview of problems encountered and potential future work
- Demonstration of working game
- Conclusions

UNIVERSITY OF LIMERICK
OLLSCOIL LUIMNIGH

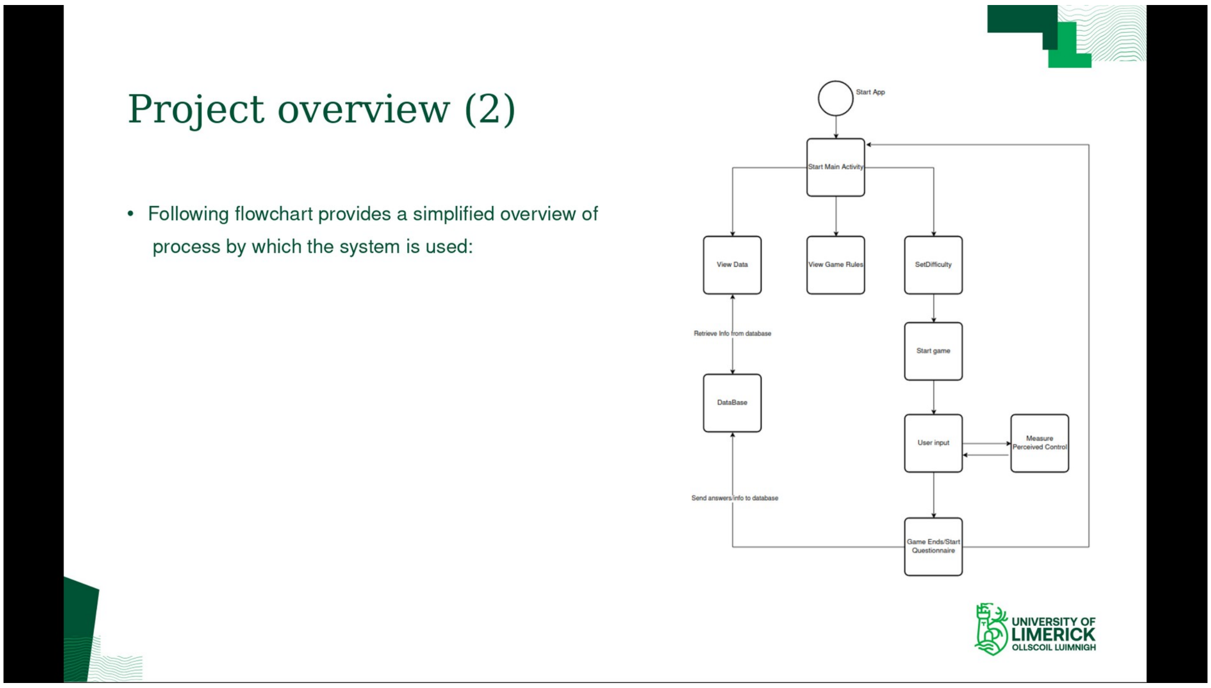


Project overview (1)

- Pupose of the project was to develop a simple game to aid in measuring a users level of perceived control.
- Game purposely will not follow random inputs – next input then marked as 'tested'

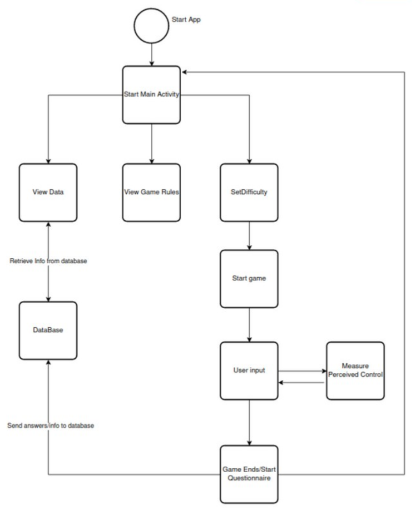
  Core gameplay involves tapping the screen to make a bird jump, hitting green clouds and avoiding blue and red clouds.
- The game functions similar to the flappy bird mobile game(with some notable differences in gameplay).
- Input data was gathered from personal testing of the game to determine potential data that would be gatherd for someone who has a deep understanding o fthe system.
- Python notebook was developed to visualise and analyse this data.

UNIVERSITY OF LIMERICK OLLSCOIL LUIMNIGH



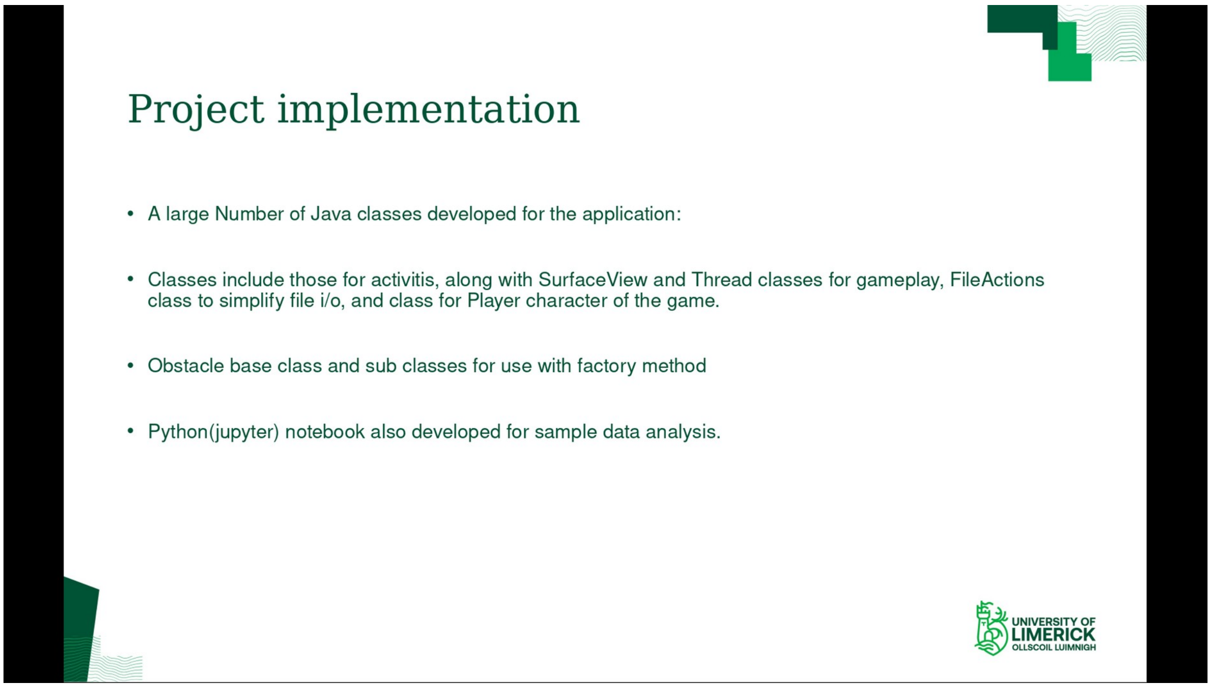
Project overview (2)

- Following flowchart provides a simplified overview of process by which the system is used:



Start App
Start Main Activity
View Data
View Game Rules
SetDifficulty
Retrieve Info from database
Start game
DataBase
User input
Measure Perceived Control
Send answers/info to database
Game Ends/Start Questionnaire

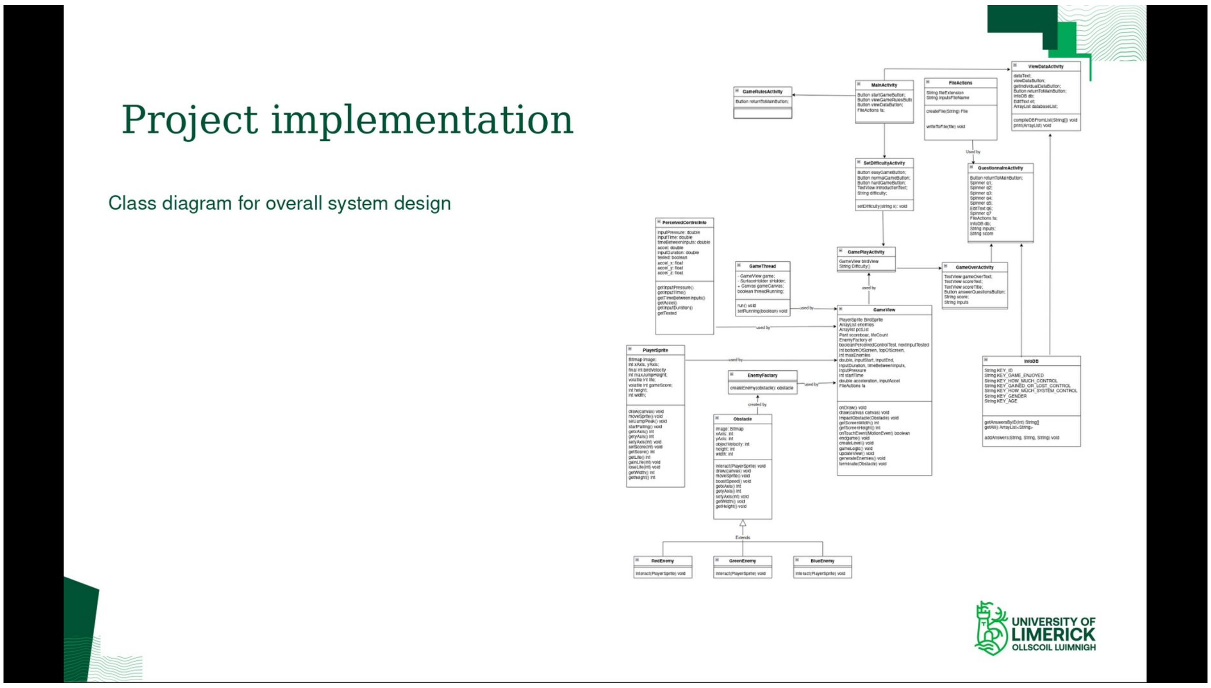
UNIVERSITY OF LIMERICK OLLSCOIL LUIMNIGH



Project implementation

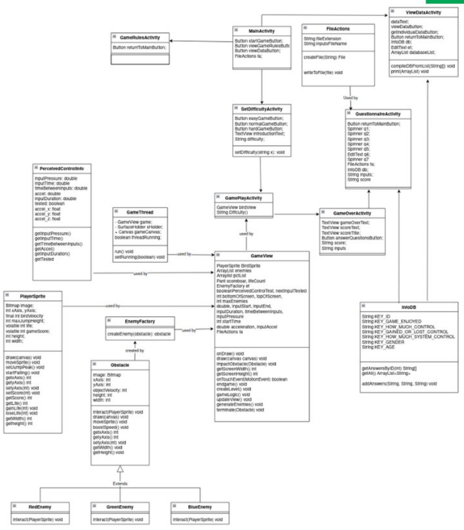
- A large Number of Java classes developed for the application:
- Classes include those for activitis, along with SurfaceView and Thread classes for gameplay, FileActions class to simplify file i/o, and class for Player character of the game.
- Obstacle base class and sub classes for use with factory method
- Python(jupyter) notebook also developed for sample data analysis.

UNIVERSITY OF LIMERICK
OLLSCOIL LUIMNIGH

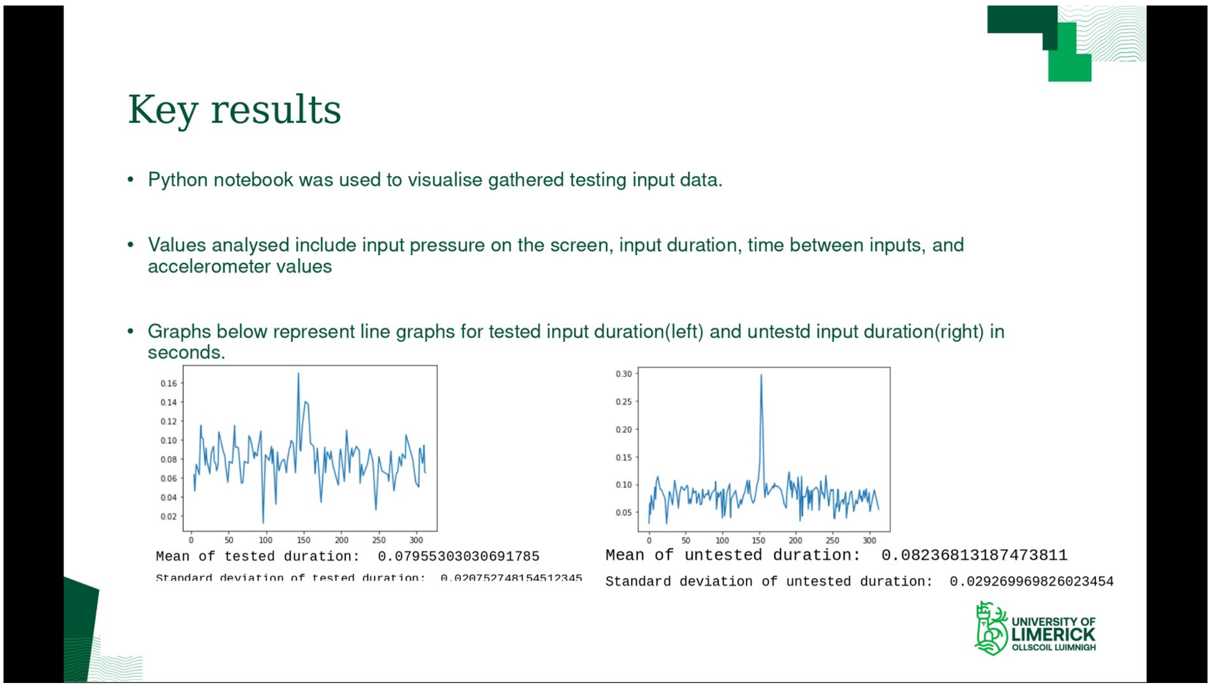


Project implementation

Class diagram for overall system design

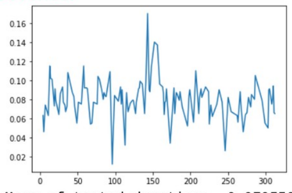


UNIVERSITY OF LIMERICK
OLLSCOIL LUIMNIGH

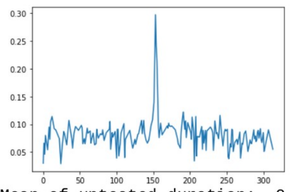


Key results

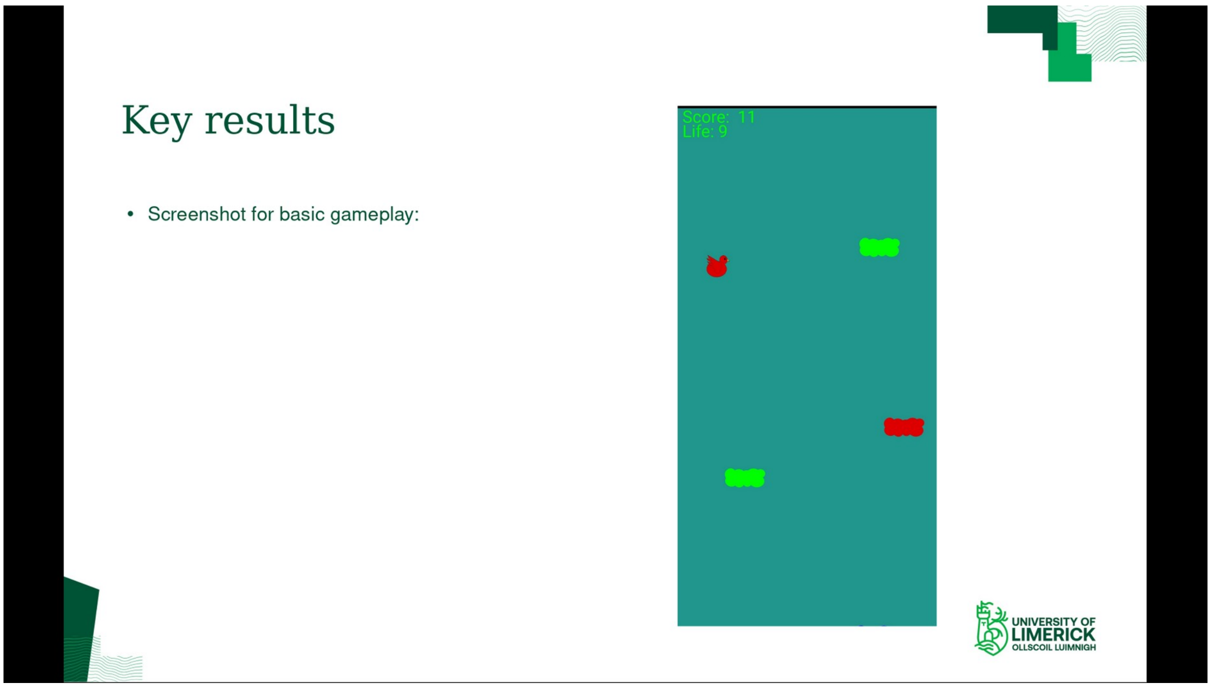
- Python notebook was used to visualise gathered testing input data.
- Values analysed include input pressure on the screen, input duration, time between inputs, and accelerometer values
- Graphs below represent line graphs for tested input duration(left) and untestd input duration(right) in seconds.



```
Mean of tested duration:  0.07955303030691785
Standard deviation of tested duration:  0.020752748154512345
```

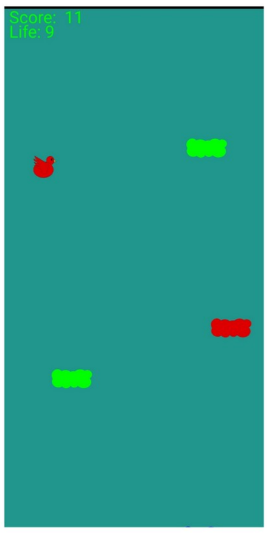


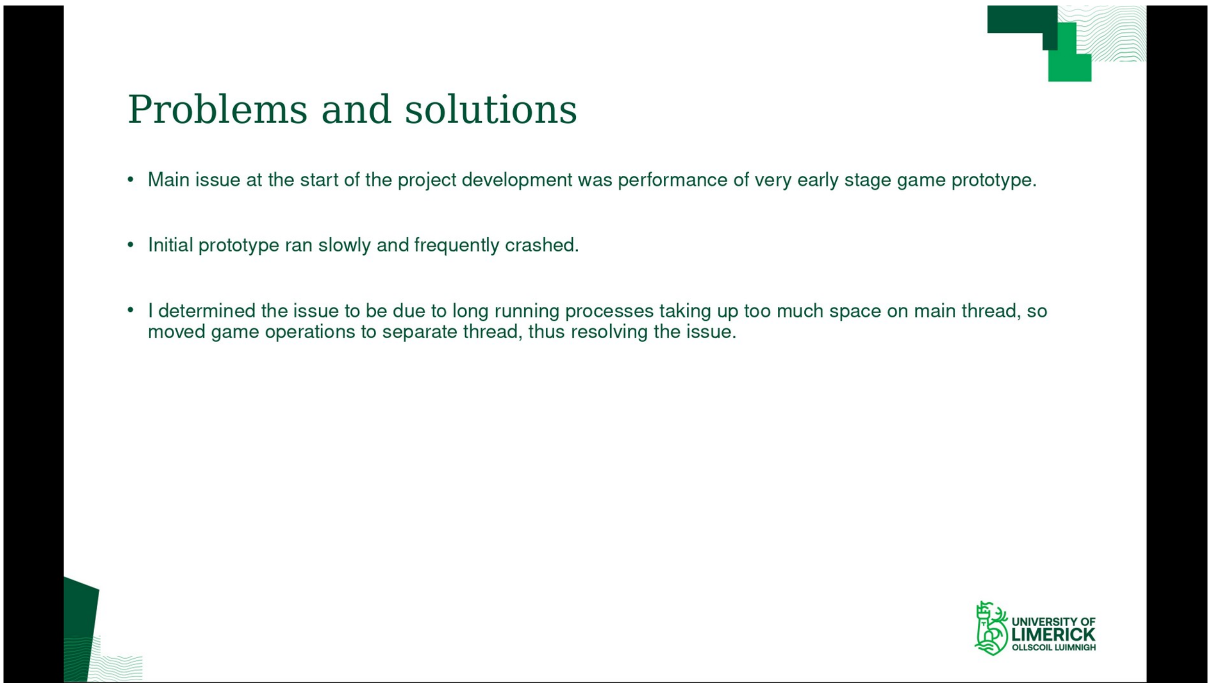
```
Mean of untested duration:  0.08236813187473811
Standard deviation of untested duration:  0.029269969826023454
```



Key results

- Screenshot for basic gameplay:

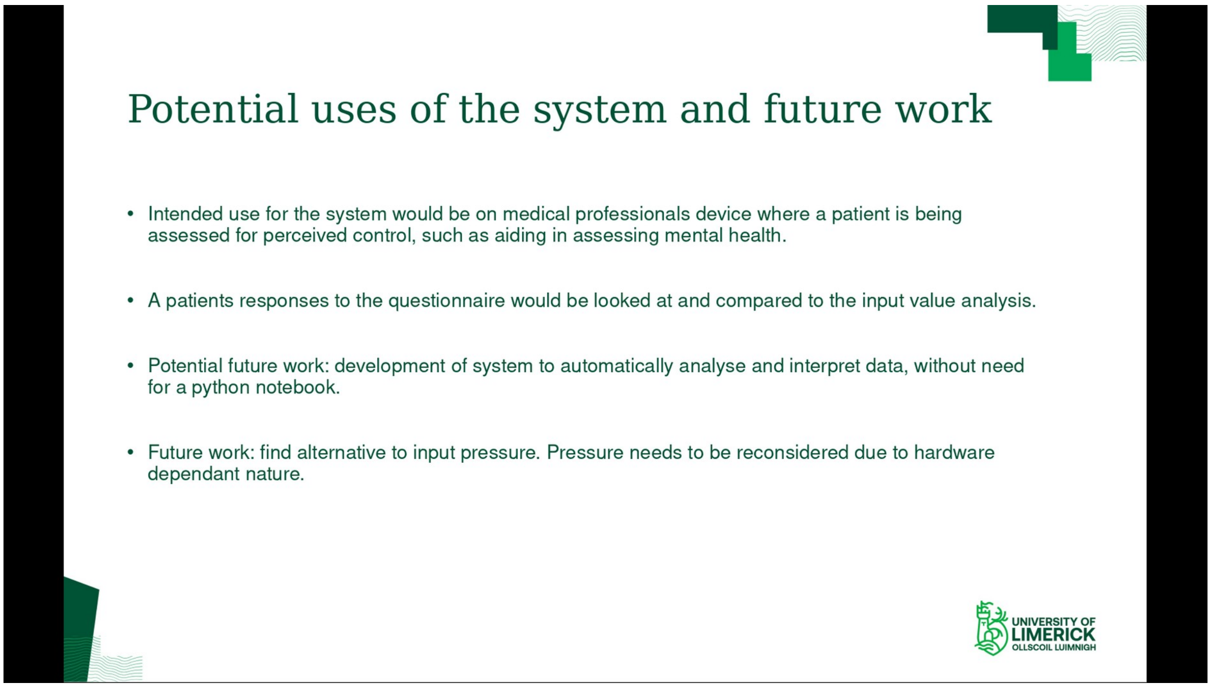




Problems and solutions

- Main issue at the start of the project development was performance of very early stage game prototype.
- Initial prototype ran slowly and frequently crashed.
- I determined the issue to be due to long running processes taking up too much space on main thread, so moved game operations to separate thread, thus resolving the issue.

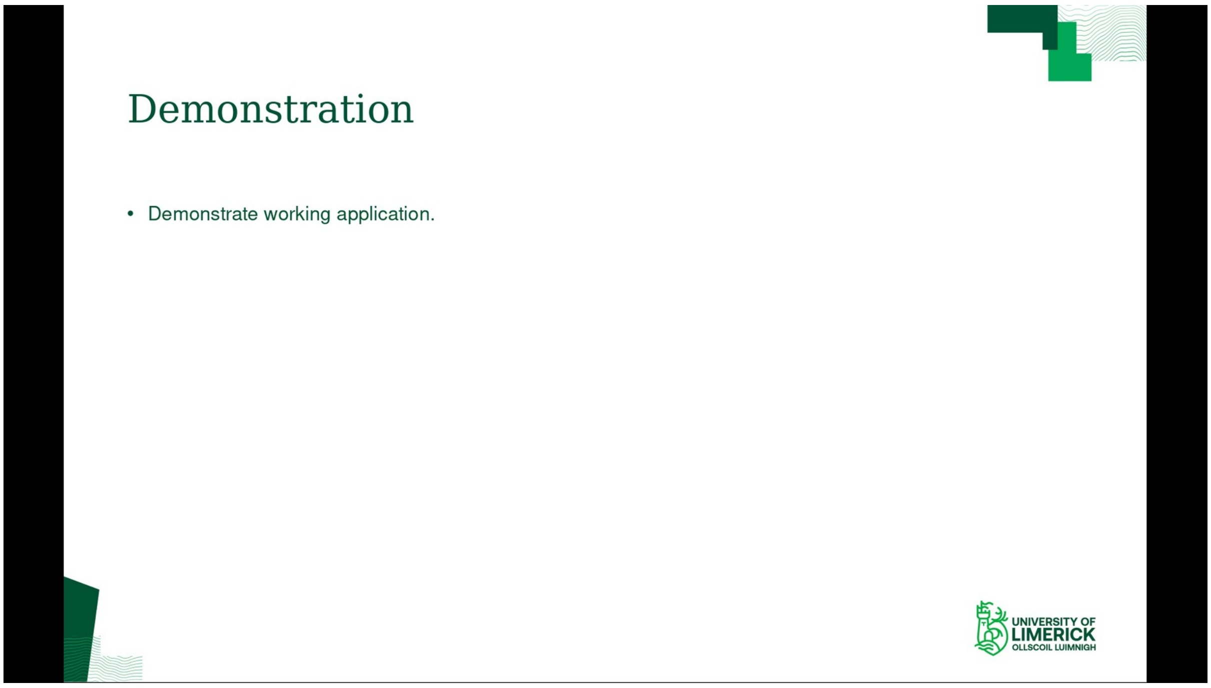
UNIVERSITY OF LIMERICK OLLSCOIL LUIMNIGH



Potential uses of the system and future work

- Intended use for the system would be on medical professionals device where a patient is being assessed for perceived control, such as aiding in assessing mental health.
- A patients responses to the questionnaire would be looked at and compared to the input value analysis.
- Potential future work: development of system to automatically analyse and interpret data, without need for a python notebook.
- Future work: find alternative to input pressure. Pressure needs to be reconsidered due to hardware dependant nature.

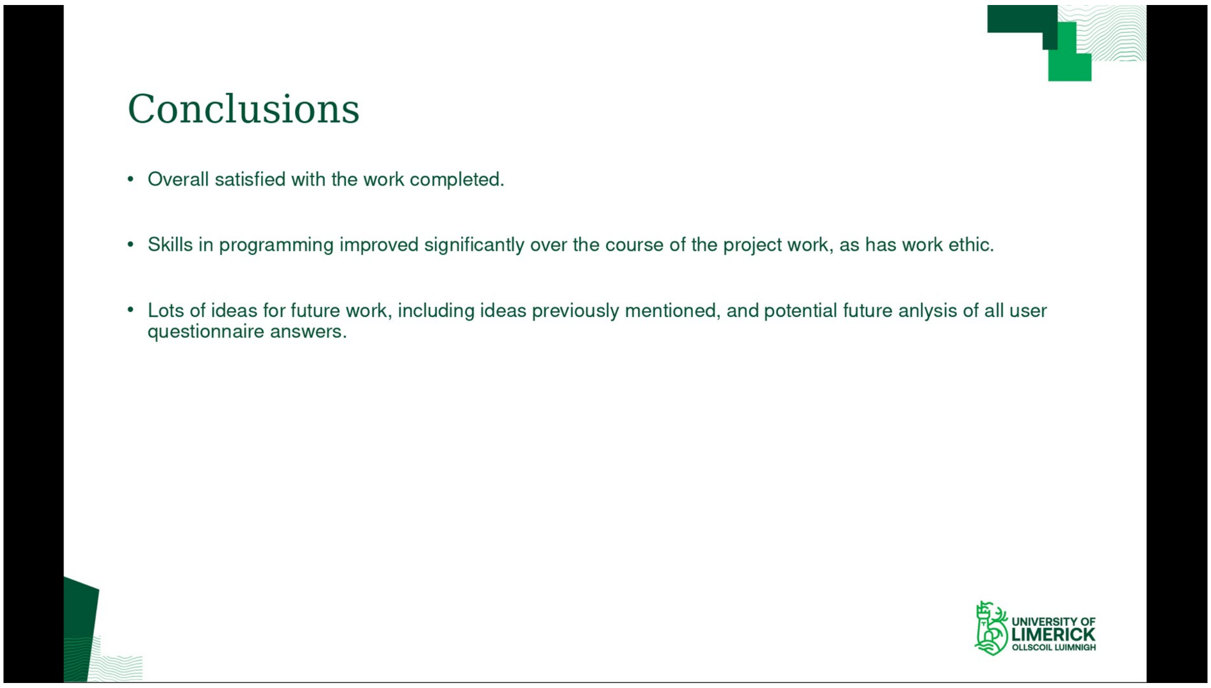
UNIVERSITY OF LIMERICK OLLSCOIL LUIMNIGH



Demonstration

- Demonstrate working application.

UNIVERSITY OF LIMERICK
OLLSCOIL LUIMNIGH



Conclusions

- Overall satisfied with the work completed.
- Skills in programming improved significantly over the course of the project work, as has work ethic.
- Lots of ideas for future work, including ideas previously mentioned, and potential future anlysis of all user questionnaire answers.

UNIVERSITY OF LIMERICK
OLLSCOIL LUIMNIGH

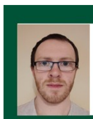
Thank you.
Any questions?

UNIVERSITY OF
LIMERICK
OLLSCOIL LUIMNIGH

University of Limerick,
Limerick, V94 T9PX,
Ireland.
Ollscoil Luimnigh,
Luimneach,
V94 T9PX, Éire.
+353 (0) 61 202020

ul.ie

Appendix C: Project poster



Android Game for Measurement of Perceived Control

Department of Electronic and Computer Engineering

Ian Rowland

Bsc Cyber Security and IT Forensics

Introduction

Perceived control is defined as the belief of an individual as to their ability to control their own actions and influence their external environment.

The purpose of this project is to develop an android game to test a users level of perceived control, both by recording user input data, and having a user undertake an in-app questionnaire after playing the game.

Python code would then be developed that can externally visualise and interpret data gathered from the game.

Aim

The Objective of this project was to develop a simple but fun game for the android operating system that would aid in the testing of a users level of perceived control.

The game will randomly, deliberately not follow a users input, and will record values associated with each input such as input pressure, input duration, time btween inputs and accelerometer values.

Method

The game was developed using android studio and the Java Programming language.

MotionEvents were used to define actions on inputs, and also used to get the input pressure value.

Input duration was measured by subtracting system time at the start of an input from system time at the endt of an input.

Time between inputs was calculated by subtracting the end time of the previous input from the start time of the current input.

Accelerometer values were calculated by getting the x y and z axis values from the android accelerometer sensor.

Determining which inputs would be tested was achieved via random number generation.

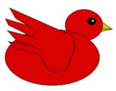
A Pandas dataframe is used in a python notebook to define input data in python code.

This allows data to be split based on whether certain rows represent tested or untested inputs, and also allows data to be split into different columns as needed

Below code defines the relevant dataframes for python code.

```
data = pd.read_csv('inputs.txt', sep =
", ", header = 0)
inputs_tested = data[data["PCT"]==True]
inputs_untested =
data[data["PCT"]==False]
colors={True:'red', False: 'blue'}
```

Image of player character used for the game



Results

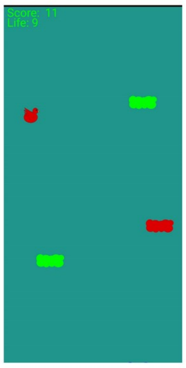
Result: Game was developed that is fully working as intended to record input data and questionnaire answers for analysis

Actual user data from other people is not gathered for the purposes of this project.

A Python notebook was also developed to demonstrate proof-of-concept interpretaton of personal testing-derived data.

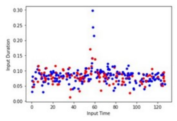
.

Screenshot of basic gameplay screen from the application

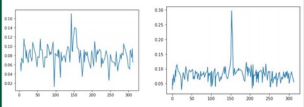


Graphs below represent how a scatterplot of plotted data might appear in a python notebook. Red datapoints would represent tested inputs, blue datapoints represent untested inputs. The plotted data was created from plotting input time against input duration for sample test data.

Plot of sample input data



Graphs below also represent difference between input duration for tested inputs(left) and untested inputs(right) for personal testing.



Line graphs for sample input durations

Conclusion and personal reflection

To conclude, I am quite satisfied with what was achieved with this project.

My skills in both the java programming language and android development were greatly improved as a result of this project.

Given either more time, or a second opportunity to conduct the project, a number of other features would have been implemented, such as features to ensure that perceived control tests do not occur too often in sequence, and a remotely accessible database

Acknowledgements

I would like to thank my project supervisor, Dr. Pepijn Van De Ven, for his support, feedback and encouragement over the course of this project.

UNIVERSITY OF LIMERICK
OLLSCOIL LUIMNIGH

Appendix D: Project progress reports

First Semester Reports:

Week 1:

For the first week of the semester, I began by establishing contact with my project supervisor, and organised my first meeting with them for week 2 of the semester. Prior to this meeting, I took consideration regarding my understanding of the project description, as well as what would need to be achieved within the project. I also considered how best to go about creating the game for the project, settling on an endless runner, developed in android studio, using the Java programming language.

Week 2:

In this week, my first meeting with my supervisor took place. We discussed the considerations made by me in the previous week regarding my project, and also discussed how best to proceed. I also begun to engage in a literature review of perceived control, where I would locate research papers and other online resources regarding perceived control. I would also locate old android development projects I had undertaken, to help in ensuring that I was fully familiar with the basic concepts of android development in anticipation for future work within the project.

Week 3:

In this week, I continued engagement in a literature review regarding perceived control. This literature review involved researching different papers and resources concerning perceived control in general, its effects on behavior, as well as perceived control in the context of a game. This Appendices 1review yielded information regarding many important topics in perceived
control, including the types of control relevant to games, as well as the concept of learned helplessness, coined by Seligman, which may be relevant to my project, in regards to inputs from users in response to a lack of expected output. I would also engage in research regarding

making simple games in android studio, to ensure that I understood what resources would be needed for the creation of a game.

Week 4:
In week 4, my second meeting with my supervisor took place. Additionally, I researched different libraries available in android studio that would be relevant to the measurement of perceived control, chiefly the MotionEvent library. This library can be used to help measure force applied to the screen, as well as time between inputs, and duration. Initially, I encountered issues when testing the motionEvent functions, where I could not get them to work properly. However, these issue were quickly rectified with further research, and watching through tutorial videos on how to properly use MotionEvents.

Week 5:
In week 5, I began the writing of my Autumn FYP Interim report. The progress made this week involved adapting my literature review for perceived control into the relevant sections of the report, as well as detailing the introduction of the report, the motivation for the report, the background theory, the action plan, and the conclusion of the report. Weekly reports and Gantt chart for intended progress are also to be included in appendices of the report, along with relevant presentation slides.

Week 6:
In week 6, I continued the writing of my autumn FYP report, and worked to complete a Gantt chart outlining the future work to be completed, and how much time I planned to allocate to all relevant tasks for my FYP. I also Developed Class diagrams, sequence Diagrams, and use case Diagrams for the relevant systems for the project. Additionally, I met with my supervisor to discuss the progress being made for the project.

Week 7:

In week 7, I continued with the final stages of the completion of my Autumn FYP report. I also met with my supervisor to critique my report, and received feedback on how best to improve my report. Additionally, I completed the presentation slides for my presentation in week 8.

Christmas Break Progress Report:
Over the course of the Christmas break, a significant amount of work was done. The relevant final art assets were created, along with the basic version of the working game, and the operations for measuring input data. The project GitHub was also updated to include these changes.

February Progress Report:
Over the course of February, a number of meetings were conducted with my supervisor. During these meetings the working game was demonstrated, and relevant feedback was provided for the code as needed. The development of the python notebook also commenced during February, along with a significant amount of work to refactor the code for the application to ensure that the full necessary functionality could be implemented. The early stages of the final report writing also commenced in late February.

March(Final) Progress Report:
During march, the final work on the project code was completed, and the majority of the work for the final report was completed, after work commenced on it in February. The poster for the project was also completed in this time.

Appendix E: Project Github

A GitHub repository was created as part of this project, as a means to back up project work in the event that work/necessary devices was lost, and to help track the progress of the project. This GitHub repository can be found at the following link:

https://github.com/ian107x/Final-year-project-Ian-Rowland