School of Electronic Engineering and Computer Science

Programme of study:
BSc FT Computer Science
4 year with IE

Final Report

GCSE Python Bingo Game

Supervisor: William Marsh
Student Name: Tirath Singh
Date: 22/04/2014

Contents

1. Introduction

1.1 Context

Is GCSE ICT putting UK students at a disadvantage? In late 2010, evidence was gathered to evaluate how ICT was being taught in schools, which led to the birth of the Computing in Schools project. The Royal Society (2012) produced a report to outline areas of failure and possible causes. Broad Curriculums, shortage of high skilled teachers, lack of teacher training and inadequate facilities in schools were the main issues highlighted.

As a result of this, the curriculum for GCSE computing was changed to equip students with relevant skills required for the industry (Gove, 2011). Teachers for exam boards such as OCR, AQA and Edexcel are now expected to have a reasonable understanding of programming and have been provided with suitable resources to teach their students.

Due to this curriculum change, there have been an increasing number of teachers who require additional materials to help their students with the basic concepts of programming. Teaching the concepts of programming through games is a popular method to keep the students engaged.

1.2 Motivation

As a recent GCSE ICT student, I am pleased to see the computer science course added. I would have appreciated some exposure to programming, to prepare me for university. Major software companies such as Microsoft and Google, who are helping to refine the computer science course, have also backed the decision to make the curriculum change.

1.3 Aim

The aim of the project is to develop a system that adapts the game Bingo, generating tickets with conditional statements in the squares rather than simple numbers. This is an implementation of an idea produced by Hazzan et al (2011). Learning programming is very challenging; however it can be made more exciting with the use of additional resources such as games.

Bingo is a game where each player is given a ticket with numbers on and they must mark off any numbers on their ticket that are called out. The winner is the player who has marked off all the numbers on their ticket first. It is believed that the game was first played in Italy, in 1530 (Snowden, 1986).

The level of difficulty can be altered to accommodate for students who require a more challenging task. The game will also require a function to allow teachers to check if a student has completed a ticket correctly.

2. Background

2.1 If Bingo

As mentioned previously, the initial concept to create a bingo game using conditional statements is from an activity by Hazzan et al (2011). The example uses 'if' statements with simple conditions and multiple variables. One difference is the language that the statements are displayed in, as I will be using Python.

Some bingo terms that will be used frequently in this project are:

- **Ticket** - This is what students should be provided with to play the game.

if x > 20 :
  print ("Bingo")
1.1
	if y > 33 :
	  print ("Bingo")
	1.4
		if z == 12 :
		  print ("Bingo")
		1.7

if x == 12 :
  print ("Bingo")
1.2
	if y == 1 :
	  print ("Bingo")
	1.5
		if z == 8 :
		  print ("Bingo")
		1.8

if x < -6 :
  print ("Bingo")
1.3
	if y == 5 :
	  print ("Bingo")
	1.6
		if z < -18 :
		  print ("Bingo")
		1.9

Figure 2.1 – Example ticket

- **Square** – Each ticket contains 9 squares. A square is referred to as true when the conditional statement equates to true.

- **Calling Card** – These are generated by the program, for the teacher to read out to players during the game. The values will satisfy a number of squares, potentially more than one square on a ticket.

x == 16;	y == 26;	z == 27;

- **Variable** – There are three variables in the example: x, y & z. A calling card gives a value to each variable.

No implementation/system was provided in the activity by Hazzan et al, so many of the algorithms required to generate tickets & calling cards will need to be formulated during the project.

2.2 GCSE Computing Curriculum

Information technology refers to the application of computer programs to solve business processes. However, computer science refers to the processes used to create usable computer programs and applications together with all theory behind those processes.

Each exam board has taken a slightly different approach to teaching/assessing computer science, however they all aim to equip students with industry relevant skills. The main areas of focus (although the level of detail varies depending on the exam board) are:

- Programming
- Computer Architecture
- Algorithms/Logic
- Binary
- Networks
- Databases

The format of assessment is:

- Written exam
- Controlled Assessment on computer that spans 15+ hours, taking place over multiple sessions.

The specific area of the curriculum concerning this project is programming. A good understanding of conditional statements is required for both the exam and controlled assessment(s).

In section 2.1.7 of the OCR GCSE specification, it indicates the student must "*understand and use selection in an algorithm (IF and CASE statements)*". There is more information in the unofficial teacher's guide (Clarkson, 2012), specifically in the Programming Project section. The use of IF statements are fundamental to the logic of the hypothetical task mentioned.

Edexcel has also released a new GCSE Computer Science qualification. In topic 2 of the specification, the level of programming knowledge is outlined however some sections are vague. It does state a preference to focus primarily on the underlying principles (logic, algorithms etc). The scheme of work for introduction and block 1(2013) is an additional document that provides teachers with a potential lesson structure for the term/year. Lesson 7 is dedicated exclusively to "Selectional constructs and relational operators", which relates to if, then & else statements.

AQA have focused on the games and mobile app sector, which they feel students can relate to more than desktop based applications that are more common in the business sector.
They offer subject-specific training before teaching the course, as well as being assigned a "Subject advisor" to provide assistance for teachers if requested.

This focus on building programming knowledge is visible in sample exam papers provided by AQA, which has a higher percentage of programming related questions in comparison to OCR. The OCR specimen paper has a broader range of questions, which will benefit students interested in other areas of IT such as networking. Edexcel has less programming questions in the written paper, focusing primarily on the underlying principles (logic, algorithms etc). Interestingly, there is more SQL programming in the Edexcel paper compared to the other 2 boards, but no other programming language is examined.

2.3 Python

The language used for the Bingo cards is Python. Languages such as Java and C++ have syntax that can be confusing to beginners (Ceder and Yergler, 2003). Python is an ideal first language for students to learn, as the syntax is very intuitive. The basic concepts for learning a programming language can be illustrated well using Python, which is sufficient for the GCSE courses.

There are many reasons why many teachers have adopted Python as a language to use for GCSE Computer Science. Below are some examples:

- **Indentation rather than curly bracket**
  For loop statements such as 'if' and 'while', the syntax is much simpler in python compared to other languages.

Python	Java

`if a < 10 :`
`  a = a + 1`
`  print(a)`
	`if(x > 0){`
	`  while(z > 10){`
	`  x = x + 5;`
	`  }`
	`}`

- **No variable declarations**
  Common programming variable types such as String and int are declared in the same way, just with the variable name. The complier will then interpret the contents of the variable suitably.

```
num1 = 7
text1 = "Hello"
```

- **Easy to understand syntax**
  Logical operators such as "and" & "or" are much easier to understand in python. Below is the same if statement given in python and java, which is clearly easier to understand in python.

Python	Java
`if (x > 5 or y == 10)`	`if (x > 5 || y == 10)`

3. Analysis

3.1 Requirements

When generating a game, there are some fundamental requirements that need to be met. A user should be able to select how many tickets they want, which should all be unique. Once tickets have been generated, calling cards are required to play the game. Calling cards must have statements that satisfy multiple squares, to ensure the game duration isn't too lengthy. Finally, the game must have at least one winner.
To make the game more complex and challenging for students, multiple aspects of the game can be enhanced. This can be achieved by changing the expressions on the tickets. The basic game will have 'if' statements with simple conditions, so either the structure of the statement can be changed (to include 'else') or the conditions can be made more complex.
Another potential enhancement is to offer additional programming languages, as some teachers may prefer to teach an alternative language.
A tool for teachers to provide feedback to students and check answers is also necessary, otherwise the students will not fully benefit from the exercise. A GUI (Teacher Control Panel) will be required for this, which can also be used to perform other tasks such as generating tickets and calling cards.

3.2 Game Generation

3.2.1 Unique tickets

A key requirement of the game is for each ticket to be unique. This can achieved multiple ways:

- Values in the square
  The obvious way to make tickets unique is to have different combinations of numbers in the squares. Each variable can have a different 'set' of values, which will then be assigned to tickets randomly.
- Structure of ticket
  The condition type of the IF statements can vary, which is also linked to the difficulty of the game. The frequency of each condition type can differ between tickets of a set.

3.2.2 Randomness

The values generated for each variable 'set' should be random, within appropriate parameters. Although it is a requirement for all tickets to be unique, there must be a framework to assign squares to tickets. This avoids undesirable scenarios such as:

- Square values that overlap on the same ticket
  E.g. **x>10** and **x>12** would both be true if **x=14** was called.
- Duplicate squares on a ticket
- Uneven distribution of variables on a ticket
  E.g. A ticket has 5 squares with **x**, but only 2 squares with **y** & **z**

3.2.3 Calling card

Each calling card will satisfy a number of statements. It is not necessary for these statements to be on separate tickets, so there can be more than one square true on a single ticket. The number of squares satisfied per calling card should be proportional to the total number of tickets, to ensure an appropriate number of rounds for each game.
The calling card values (number of squares which equate to true) will increase/decrease proportionally depending on the number of tickets. Therefore in most cases, the number of calling cards should remain in the desired range.

3.2.4 Game Winner

Another key requirement of the game is to have at least one ticket completed (all squares equate to true). However it is not a requirement to have only one unique winner, which is tantamount to the original bingo game.

3.3 Game Attributes

3.3.1 Level of difficulty

One aspect of the game that has large potential for development is the level of difficulty. This is related to the complexity of the code within each square. There are two major areas that can be developed:

- Structure of expression
  The basic structure of the expression is a single 'if' statement. The natural progression from this is to include elif or else statements. Simple assignment operations could also be included, prior to the conditional statement. Nested loops can be used to add complexity.

```
if x > 15 :
    print("Bingo!")
```

```
if x > 15 :
    print("Bingo!")
elif x == 6 :
    print("Bingo!")
```

```
y = x
while y < 2 :
    print("Bingo!")
```

```
while y < 2 :
    if x > 15 :
        print("Bingo!")
```

- Conditional statement
  Similarly for conditional statements, there are multiple approaches to increasing the difficulty. Potential steps would be to include and/or clauses, as well as not equals (!=) and other comparison operators.

```
if x == 15 and y > 11 :
    print("Bingo!")
```

```
if x == 15 or x == 7 :
    print("Bingo!")
```

```
if x <= 15 or x != 7 :
    print("Bingo!")
```

3.3.2 Alternative Programming Languages

As there is currently no universal programming language for GCSE, additional programming languages could be made available to appeal to a larger audience. Java and C++ are popular Object Oriented languages and sought after by most employers.

```
if(x == 7){
    System.out.println("Bingo!");
}
```

```
if(x == 7)
    cout << "Bingo!" << endl;
```

3.4 Teacher Control Panel

3.4.1 Generate tickets

The initial aim of users will be to generate tickets, so the interface needs to be easy to understand for first time users. The variables for creating a new game are:

- Number of Tickets
- Difficulty
- Programming Language

3.4.2 Generate calling cards

Playing the game requires two core functionalities: generating a calling card and checking how many squares should be ticked on any ticket. The main window should display calling card values, as well as other meta data such as how many cards have been generated.
The teacher should be able to generate as many calling cards as they wish. By letting the teacher decide when to generate a calling card, this allows them to control the time delay between rounds.

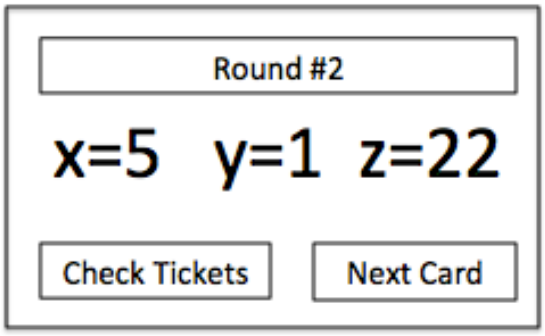


Figure 3.1 – Screen design for generating and displaying calling cards

3.4.3 Check status of tickets

In the main window, there should also be an option to view the status of tickets, to check the progress of students. This just needs to be a clear representation of any given ticket, indicating which squares should be ticked. Other game functionality such as generating a calling card should only be available in the main window.

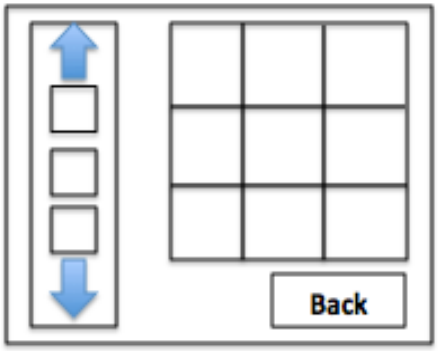


Figure 3.2 – Screen design to view tickets

3.5 Feedback

At any point during the game, the teacher should be able to check a specific ticket to see if the student has completed their ticket correctly. This can be achieved using the "Check Tickets" window.
However, this does not specifically identify where a student went wrong if they ticked an incorrect square. This would require frequent supervision by the teacher(s).

Additional data could be included on the main window to assist the teacher if they want to know approximately how many squares should be ticked:

- The least number of squares ticked on a ticket
- The most number of squares ticked on a ticket
- The number of tickets with all squares completed

These could be displayed without labels, to avoid students taking advantage of them.

3.6 Instructions

Instructions for the game should be accessible before generation and during gameplay. The instructions will explain the basic game mechanics and how to use the software correctly.

- **What does New, Save & Load mean?**
  A brief description of the functions should be displayed when starting the application.
  - New Game – Choose how many tickets you want and generate a PDF to start playing.
  - Save Game – If you want to save a set of tickets for future use.
  - Load Game – Start playing with previously generated tickets.

- **How to generate PDF tickets?**
  Help should be available on the menu bar to provide more detailed instructions when starting a new game. This should include where the pdf document is saved.

- **How to start playing the game?**
  This should include how to create calling cards and general information about the game window.

- **How to check a ticket during the game?**
  The current check status window is self-explanatory, however additional features may be added which may require explanation.

4. Design & Implementation

This chapter will outline the high-level design and architecture of the project, based on the requirements produced. It will then focus on how the game is generated and game algorithms used. Finally, a description of the teacher control panel has been added, including a user guide.

4.1 Architecture

The MVC design pattern was chosen for this project as the game dynamics suit this structure well. Below, figure 4.1 outlines the key components of the game and which package they belong to. The classes within the controller package retrieve information from the relevant view class (depending on user actions) and update the model classes. This is required when the user selects key game attributes using the teacher control panel, such as the number of tickets to generate.

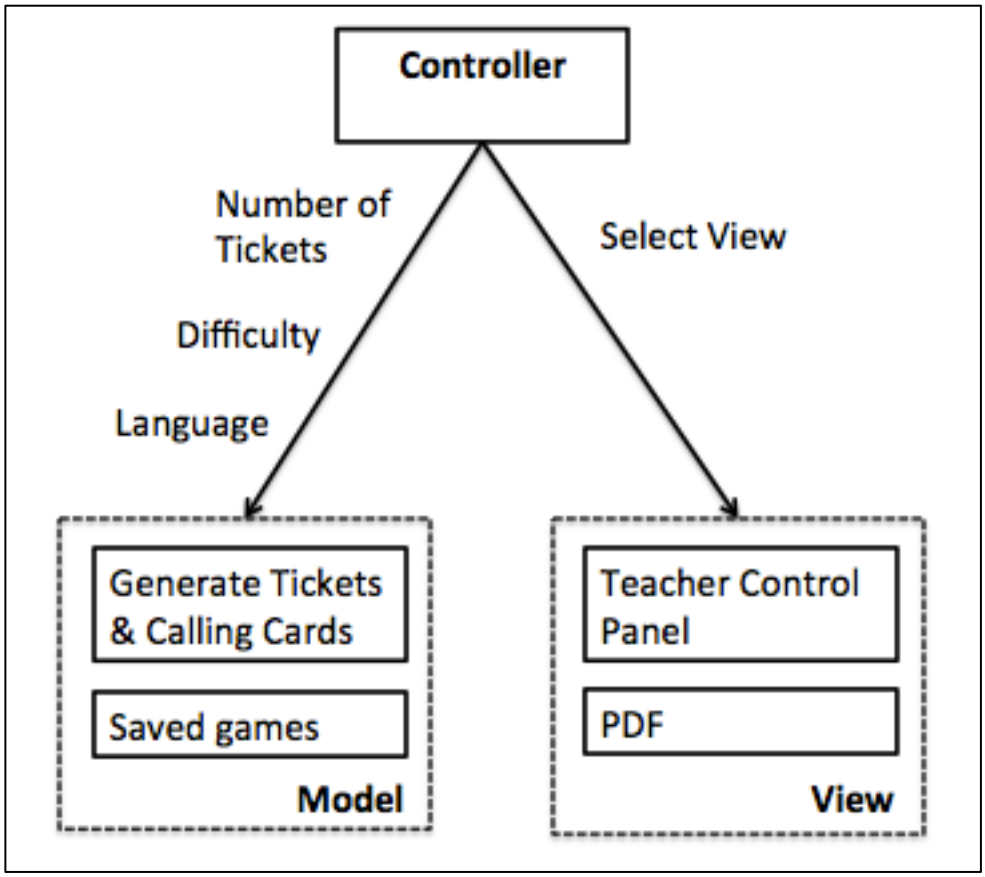


Figure 4.1 – Package design

4.1.1 Model Package

The high-level class diagram for the Model package is shown in Figure 4.2, to show how classes within the Model package interact with each other. This is followed by an overview of each class, pointing out key methods used.

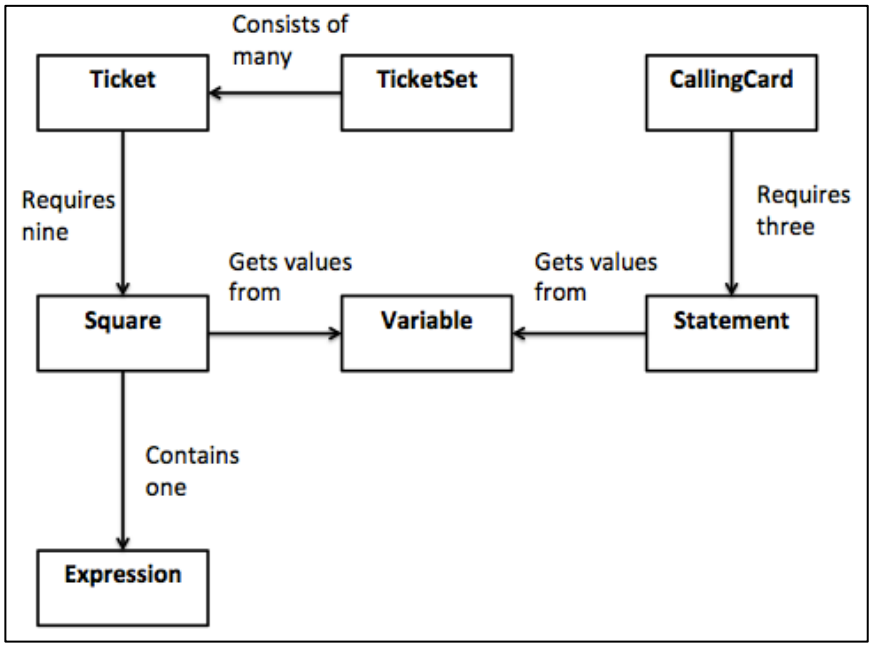


Figure 4.2 – Model package

The variable class is key for both Ticket & Calling card generation. Each instance of the Variable class contains numbers that the Square class uses to display an expression. The Statement class, used to display values on the Calling card, then manipulates the numbers from the Variable class.

CallingCard
The CallingCard class gets the count of a random value from all Statement instances and checks if the sum of these counts is within a suitable range. The count is simply the number of squares that contain a given value. The range is dependent on the number of tickets generated.

Expression
The Expression class gets information from the Square class and combines it to a single string. The structure of the text is dependent on the type of conditional statement and programming language.

Square
Each instance of the Square class gets a value from a Variable, depending on which parameters were passed in the constructor of the Square. It then instantiates an Expression, passing on the relevant information.

Statement
There is a Statement class for each Variable class. The list of values from the Variable class is used to produce random statements relevant for the game (at least one square will be true). The Statement class also works out how many squares equate to true for each value in the Variable lists of values.

Ticket
The Ticket class instantiates nine squares, however the type of squares will differ depending on the difficulty selected. Advanced games will have squares with more complex conditional statements.

TicketSet
The TicketSet class generates the required number of Ticket instances for the game. It also controls the structure of each ticket (the number of greater than, less than & equals statements per variable).

Variable
Three Variables are initialized at the start of game, once the game attributes have been selected. Each variable has a separate set of arrays containing numbers to use for tickets/calling cards. The variable class also sets a limit of how many times each value can be assigned to a ticket. As mentioned previously, both the Ticket and Calling Card class uses the arrays of values.

4.2 Game Generation

The controller package catches all the user actions from the Teacher Control Panel and responds by making changes to the Model package accordingly. Initially, the view is only required to gather game attributes from the user. The remaining functions of the view are used for playing the game, explained in the Teacher Control Panel section later.

Figure 4.3 shows the key steps to generate and complete a game.

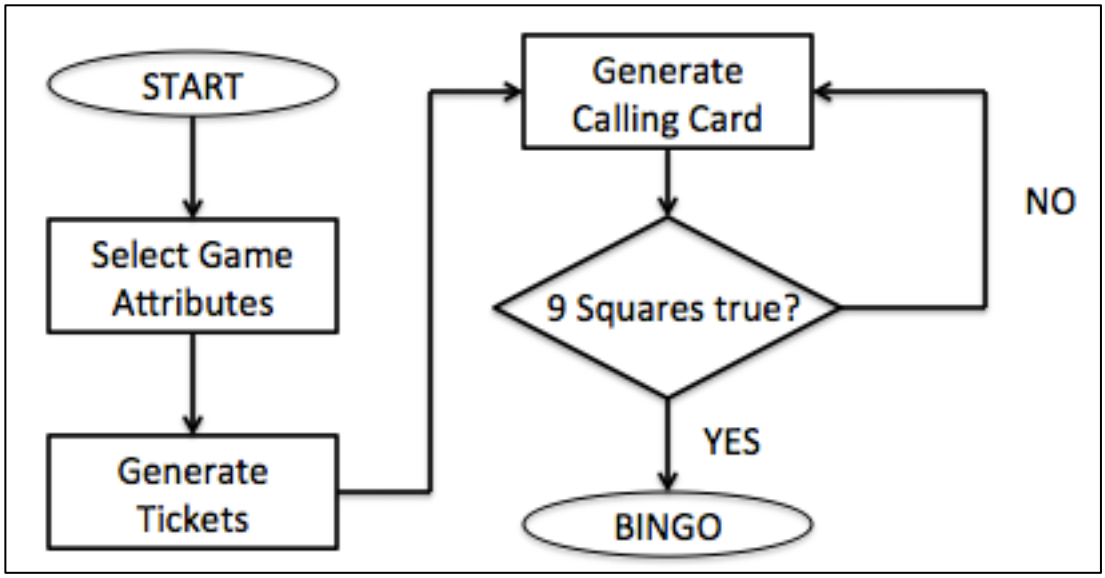


Figure 4.3 Flow diagram of game generation

4.2.1 Select Game Attributes

Before generating tickets, some game attributes must be specified by the user. These values are passed on to the relevant classes in the Model package. Below, figure 4.4 shows the game attributes.

Figure 4.4 – Selecting game attributes

4.2.2 Generate Tickets

Once the game attributes have been selected, the values are passed to the constructor of the Generate class in the controller package. The Generate class initializes the three variables in the game, as well as the TicketSet class. As values on the tickets need to align with lists in the variable, the variables are passed as a parameter in the TicketSet constructor.
The TicketSet class defines how many times a values (such as x == 7) can be repeated, depending on how many tickets need to be generated. The structure of each ticket is then defined, with the possible layouts for each variable shown in figure 4.5.

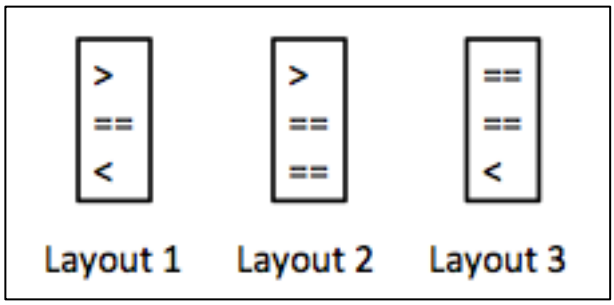


Figure 4.5 – Possible layouts for each variable in a ticket

There is a method to make sure the same 2 equals' statements aren't on the same ticket at any point. The previous equals value is passed to the method, to avoid duplication on a ticket.
There is also a scenario where the last ticket generated contains 2 equals' statements with the same value. A check has been put in place to ensure if this happens, the TicketSet must be re-generated to avoid an infinite loop.

4.2.3 Generate Calling Card

In order to generate a calling card, three statements must be supplied. Each Statement represents one variable instance. The Generate class initializes the Statement classes, which combine to make each CallingCard instance.

When selecting which Statements to use for a CallingCard, a random value is selected for each statement. The count (number of tickets which this "value" appears on) is then worked out by checking each ticket. The sum of the three statements is calculated and if it falls within a specified range, the calling card is generated. If the sum is too small or too large, new values are selected. Figure 4.6 is a visual representation of how statements are combined to produce a CallingCard.

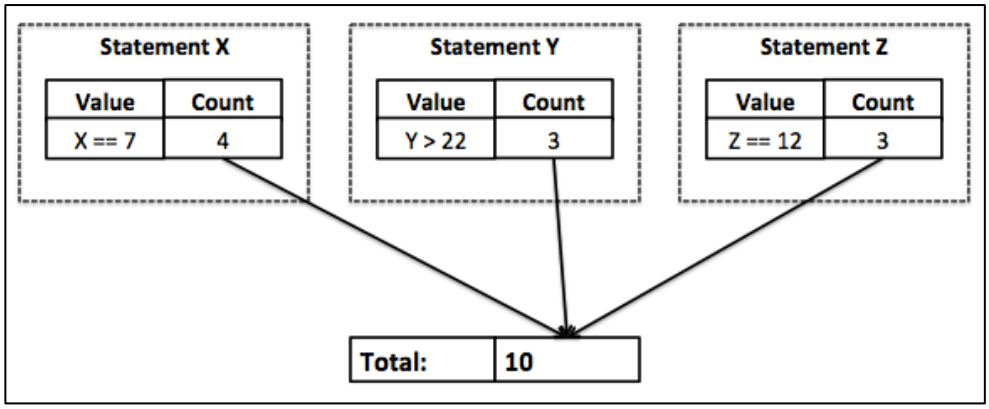


Figure 4.6 – Logic for selecting statements on a calling card

After each Calling Card is generated, each ticket must be checked to see if all nine squares have been completed (i.e. bingo!). This is also required for the "Check Ticket" window in the Teacher Control Panel, which will be explained in the Teacher Control Panel section.

4.2.4 Output to PDF

Once the TicketSet has been generated, the tickets must be output to a PDF document. The library used is iText, an open source library to create and manipulate PDF Documents. The name of the PDF generated is specified when selecting game attributes at the start of the game. The steps to generate a PDF containing multiple tickets are shown in figure 4.7 below.

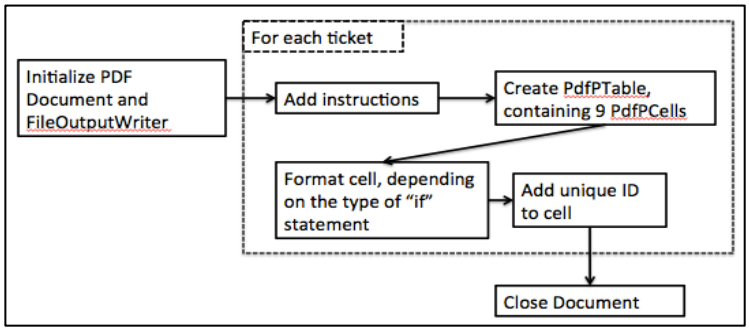


Figure 4.7 – Steps to generate PDF containing tickets

Each page has 2 tickets, where a ticket is represented by:

- Short description of the rules
- PdfPTable - an object in the iText library containing 9 PdfPCells.

As the size of cell will change depending on the contents, a method *formatCell()* is called to ensure exactly 2 tickets fit on a page. For an advanced game, most squares have more code than they would if it was a basic game, see figure 4.8 below.

Basic	Advanced

if x > 15 :
  print("Bingo!")
	if x > 15 :
	  print("Bingo!")
	elif x == 6 :
	  print("Bingo!")

Figure 4.8 – Comparison of cell size between basic and advanced game

As a result of this, the formatting must be adjusted depending on the game attributes, shown in figure 4.9 below. The "lang" variable is a Boolean, which is true if the user has selected to play in Java. The font size of a cell changes, if the expression contains an "if & else" statement or a nested "if" statement.

```
boolean b1 = elif[num];
boolean b2 = nested[num];
if(!lang){
    if(b1){
        content = new Phrase(tickets[num]   +"\n"+ "   print (\"No Bingo!\")" + "\n"+ "else :" + "\n"+ "   print (\"Bingo!\")",
        cellID = new Phrase(ticketNum + "." +id, font4);
    }else if(b2){
        content = new Phrase(tickets[num]   +"\n"+ "      print (\"Bingo\")" + "\n", font2);
        cellID = new Phrase(ticketNum + "." +id, font4);
    }else{
        content = new Phrase(tickets[num]   +"\n"+ "   print (\"Bingo\")", font1);
        cellID = new Phrase("\n"+ ticketNum + "." +id, font4);
    }
}else{
    if(b1){
        content = new Phrase(tickets[num]   +"\n"+ "   System.out.println(\"No Bingo!\");" + "\n"+ "} else {" + "\n"+ "   System
        cellID = new Phrase(ticketNum + "." +id, font5);
    }else if(b2){
        content = new Phrase(tickets[num]   +"\n"+ "      System.out.println(\"Bingo\");" + "\n" +"   }" +"\n" +"}", font5);
        cellID = new Phrase(ticketNum + "." +id, font5);
    }else{
        content = new Phrase(tickets[num]   +"\n"+ "   System.out.println(\"Bingo\");" +"\n" +"}", font3);
        cellID = new Phrase(ticketNum + "." +id, font4);
    }
}
```

Figure 4.9 – Section of code from formatCell() method

4.3 Game Algorithms

The application requires some complex algorithms to meet certain requirements of the game. This section will walk through a few of the key algorithms used by the application.

4.3.1 Get Square value

Each square instance must get a random value from a variable list. The variable class has a separate method for each list (greater, equals & less), each corresponding to a different list of values. In this example, the square instance must retrieve a value from the greater list, using the method getGreater() shown in figure 4.7.

```
public int getGreater(){
    int limit;
    if(mult==1){
        limit =1;
    }else{
        limit=mult;
    }
    int num = randInt(0,gr.size()-1);
    Iterator i = gr.entrySet().iterator();
    if(i.hasNext()) {
        Map.Entry<Integer, Integer> pairs = getPos(i, num);
        int val = pairs.getValue();
        while(val == limit){
            num = randInt(0,gr.size()-1);
            i = gr.entrySet().iterator();
            pairs = getPos(i, num);
            val = pairs.getValue();
        }
        val++;
        gr.put(pairs.getKey(), val);
        return pairs.getKey();
    }
    return 0;
}
```

Figure 4.7 – Code to get random value from greater arraylist

The "limit" is the number of times a value can be displayed on separate tickets. An iterator is required to cycle through the HashMap "gr", unlike an arraylist which can easily be cycled through using a "for" loop. To get a random value from the HashMap, getPos() was implemented (see Figure 4.8). This method returns the key & value from the random position in the HashMap. The value in the HashMap is the count, whilst the key is the actual value to be displayed on the square. If the value (count) is equal the limit, a new value needs to be selected. Once a value is found which is below the limit, it is returned.

```
public static Map.Entry<Integer, Integer> getPos(Iterator i, int num){
    int counter = 0;
    Map.Entry<Integer, Integer> pairs = (Map.Entry)i.next();
    while(counter!=num && i.hasNext()){
        pairs = (Map.Entry)i.next();
        counter++;
    }
    return pairs;
}
```

Figure 4.8 – Code to get random position in iterator

4.3.2 Check for duplicate squares on final ticket

To avoid being stuck in an infinite loop or displaying duplicate squares on the final ticket of a game, a check was implemented. The aim of the check is to go through the equals list for each variable to make sure that one value does not have (limit-2) as the count. This would imply that all other values have reached the limit and the remaining 2 squares would be duplicate values.

```
if(i==ticketNum-1){ //Last card
    HashMap<Integer, Integer> xMap = x.getEqualsMap();
    Iterator i1 = xMap.entrySet().iterator();
    Map.Entry<Integer, Integer> pairs = null;
    for(int j=0; j<xMap.size();j++) {
        pairs = (Map.Entry)i1.next();
        int val = pairs.getValue();
        if(val ==(limit-2)){
            System.out.println("2 of the same expressions for last card X. New cards to be generated");
            return false;
        }
    }
```

Figure 4.9 – Code to check for duplicate values

4.3.3 Generate Calling Card

The steps to generate a calling card have been outlined in Figure 4.10 below.

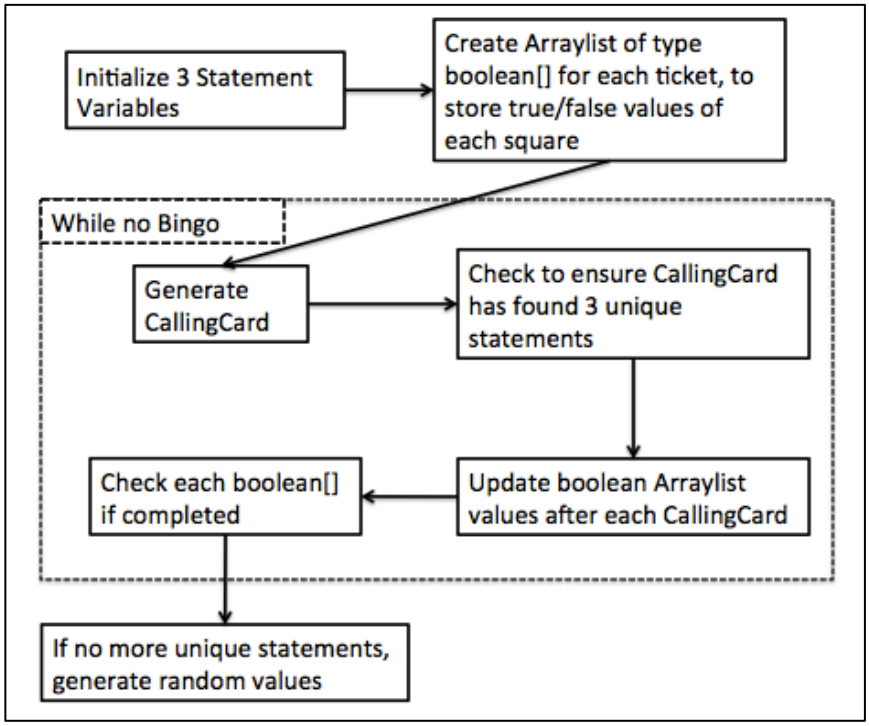


Figure 4.10 – Steps to generating Calling Cards

As previously stated, a CallingCard is composed of 3 Statements. Therefor, the statements must be initialized before generating any CallingCard. Each Statement gets the "value" HashMaps from a unique Variable. The maps are converted to TreeMap, which can then be sorted. This is useful when getting the count of a particular value in the greater & less maps. As values overlap, then it makes working out the count much more efficient.

In order to know when a ticket has been completed (all 9 squares equate to true), an arraylist containing boolean arrays (of length 9) was created. All values are set to false initially. Whilst no position in the arraylist has an array of 9 true values, CallingCards are generated in a loop.

A CallingCard can attempt combinations of statements up to 1000 times, before all the CallingCards must be removed and re-generated if there is no true array (i.e. no bingo). If at least one ticket has been completed, CallingCards can still be generated even if there aren't any unique Statements remaining. This allows the teacher to continue playing after a student has shouted bingo.

4.4 Teacher Control Panel

The teacher control panel was designed to make it easier for the teacher to play the game with students. It provides the teacher with the ability to customize game attributes, such as the difficulty of conditions within squares.

A key feature for the teachers is the ability to check the progress of tickets at any point. Checking tickets without this feature takes a lot of time, so the teacher may have only checked tickets at the end of the game, making it harder to identify any mistakes a student made. With regular checking using the feature, it is easier to identify where the student went wrong.
The logic used to implement this feature is similar to that used to check for bingo after a calling card is generated. The interface was designed to be simple and easy to use.

Another feature is the ability to save tickets and load them for future use. Once the teacher has generated tickets, they may not want to use them immediately with a class. A teacher may also want to use the same ticket set multiple times, with different calling card sequences. To conform to the save functionality most users will be familiar using, the program requests a name to identify a game.

4.4.1 User Guide

The following user guide was provided to teachers interested in testing the software. The guide explains the main functionality of the application, as well as how to install it (which is not required here).

How to start a new game

When you load the game you will see the screen below. It describes what you can do initially. To start a new game, select file from the menu bar and click New Game.

Figure 4.11 – Start screen

You will then get a pop-up to select the name of the PDF and how many tickets you want. When naming the PDF, make sure to keep the file extension.
Don't get confused between the name of the PDF and the name of the save game, they are not related! This has not saved the game; it's just generated the PDF document.

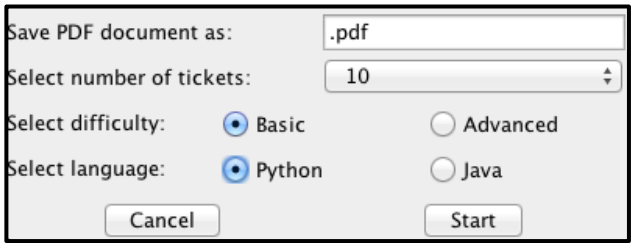


Figure 4.12 – Game attributes screen

How to save a game

If you want to re-use a set of tickets for future use, you should save the game. This is on the menu bar shown below.

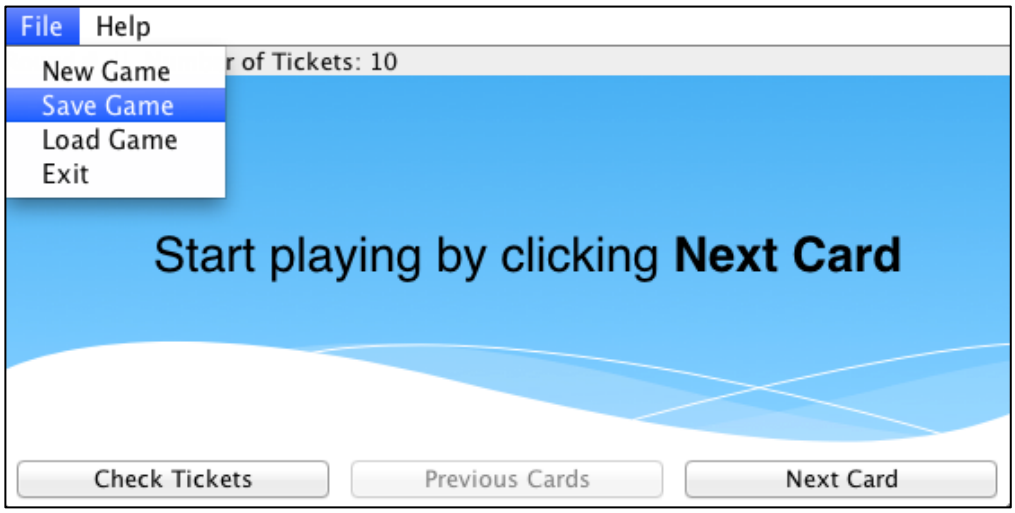


Figure 4.13 – How to save game

By saving a game, the game will remember what tickets you have, but will generate new calling cards each time you load it.

Use appropriate names when saving, to make life easier when you decide to load the game.

How to load a game

You can load a previously saved game when starting the application, or whilst playing a different game.

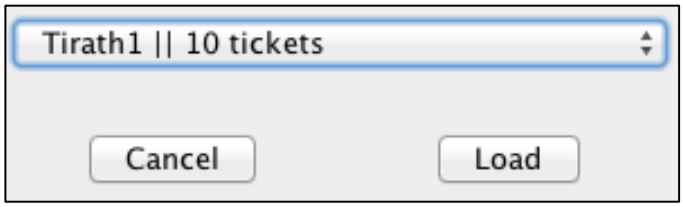


Figure 4.14 – Load window

If you are playing a game and wish to switch games, the current game will be lost unless saved.

If you load the same game multiple times, you will get different calling cards. This allows you to re-use the same tickets as many times as you wish.

How to play the game

Start playing by creating calling cards. One calling card is generated per round. The number of rounds is shown at the top left corner of the window.

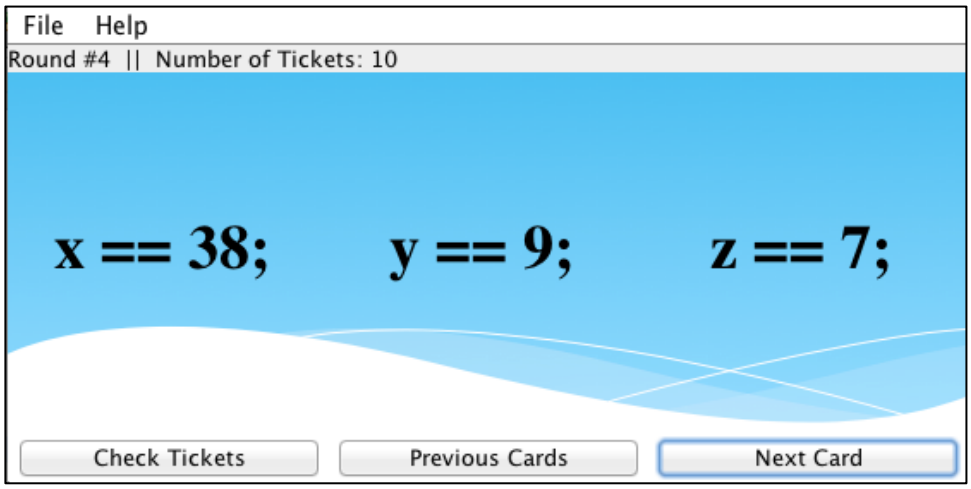


Figure 4.15 – Game window, with Calling Cards generated

At any point during the game, you can check the status of any ticket. This is particularly useful when a player claims to have completed a ticket and you wish to verify this quickly.

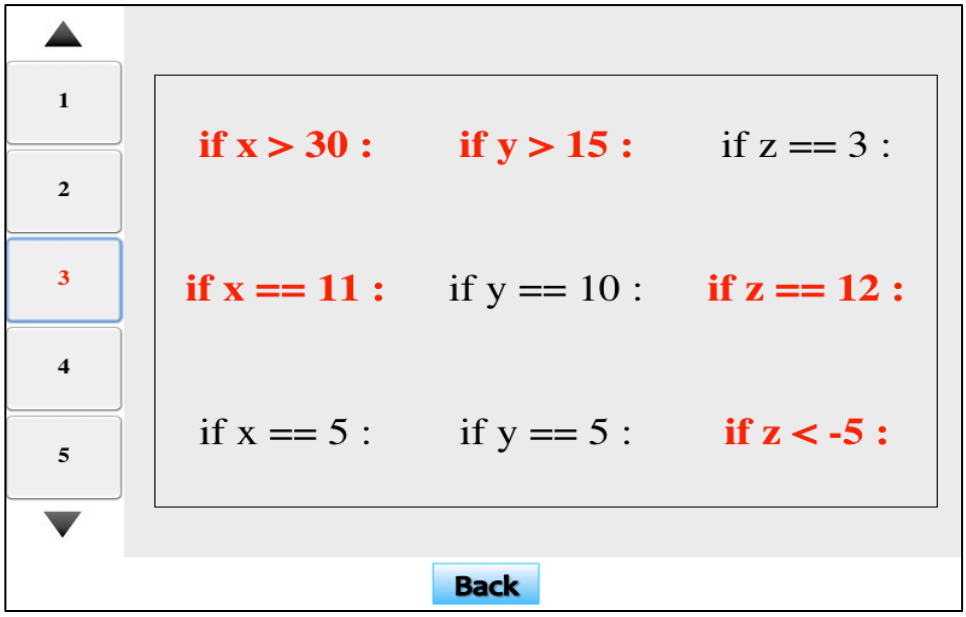


Figure 4.16 – Check ticket window

On the left hand side are ticket numbers, and you can scroll using the up & down arrows. On the example shown above, ticket 3 has 5 squares completed (in red) and 4 squares that have not been called yet.

5. Evaluation

5.1 Ethics approval

Before sharing information with teachers, the material had to be approved by the Research Ethics Committee. As well as supplying any material that would be sent to participants (teachers), a form to determine the nature of the participants had to be completed. Although teachers may decide to test the game out on children, it was not a requirement of the research and we do not put any pressure on teachers to test it on their students. We simply require their professional opinion on the game.
A copy of the approval from the Research Ethics Committee has been added to the appendix.

5.2 Advertising to teachers

Once the ethics form was approved, a handful of teachers who showed interest in the project were contacted via email. These teachers were provided with an early version of the game. This included:

- PDF containing 10 tickets with basic difficulty
- PowerPoint Presentation with basic game rules
- Excel Spreadsheet to show the progress of each ticket after each calling card (shown in figure 5.1)
- Participation document containing details of the study

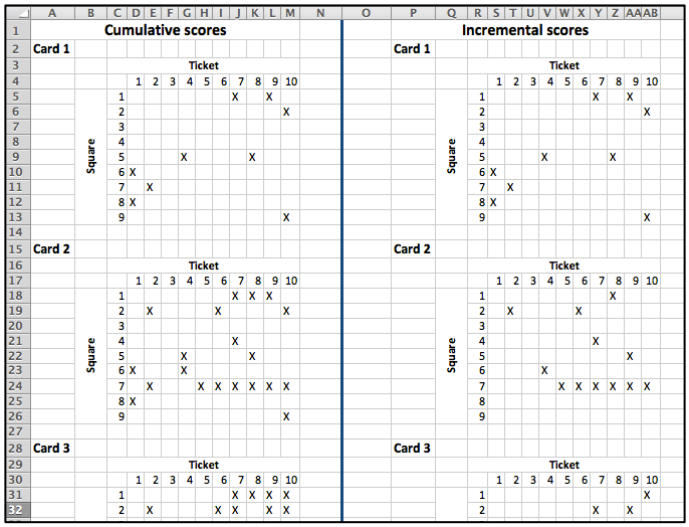


Figure 5.1 - Part of the answers spreadsheet

To advertise to a larger audience, details of the project were posted on the CAS (Computing at school) website. The website has a section where teachers can upload useful resources and other teachers can like or post comments.

5.2.1 Website

To make it easier to download the game and gather all relevant resources in one place, a website was created. The site is split into 4 sections, described below.

The aim of the "About" page is to give site visitors a brief overview of the project, as well as a short description of what to expect on other tabs. A link to the user guide is also provided.

Figure 5.2 – Screenshot of "About" webpage

The "Work" page contains links to each version of the game, along with a short description.

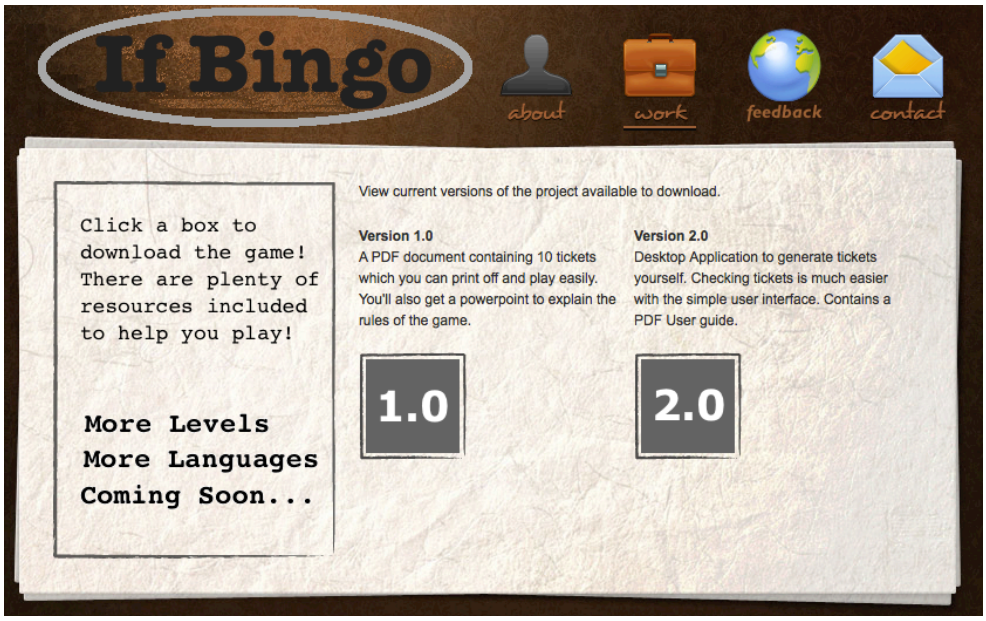


Figure 5.3 – Screenshot of "Work" webpage

The SurveyMonkey questionnaire has been embedded on the feedback page, to make it quicker and easier for teacher to fill out.

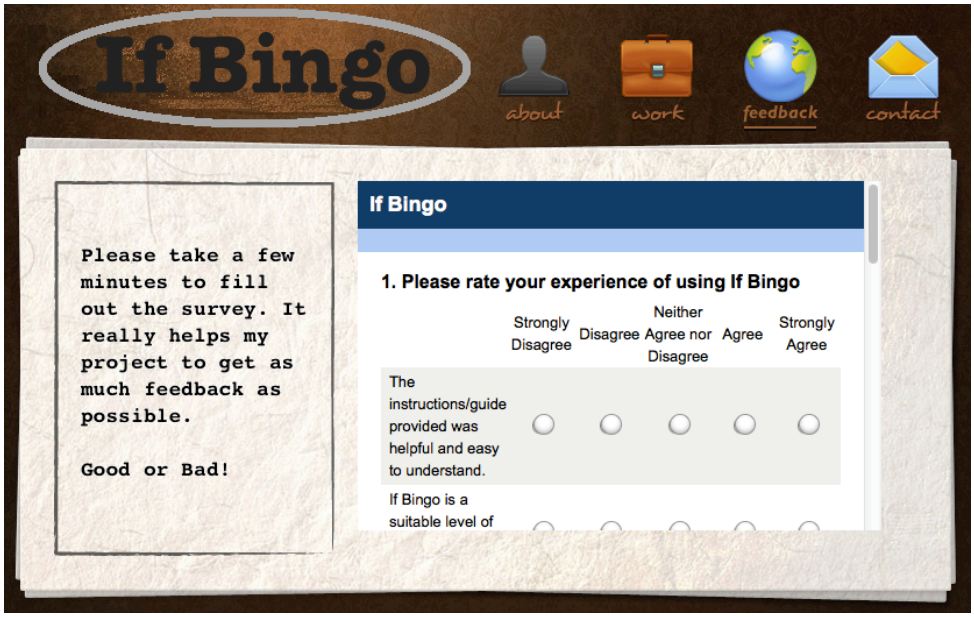


Figure 5.4 – Screenshot of "Feedback" webpage

Finally, a page with contact details and encouraging users to report bugs if they come across any.

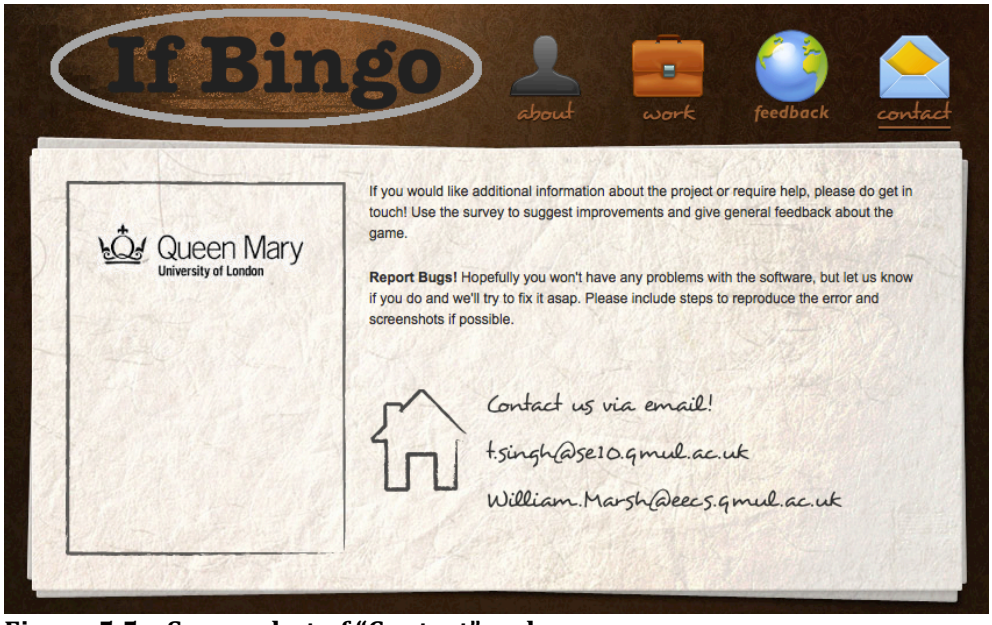


Figure 5.5 – Screenshot of "Contact" webpage

5.3 Questionnaire results

Three teachers provided feedback of the first increment of the game, which has been included in the appendix. The feedback was extremely positive and also identified a few areas that could be improved in the next increment.

One teacher found the game the game to be engaging for the students and a suitable level of difficulty. However, they struggled to use the Excel "answer" sheet, which caused some confusion for the teacher and players.

One of the other teachers also commented on the difficulty of the game. Although they believed the students would definitely find the game engaging, they suggested making the statements more complex or focusing specifically on one topic.

The multiple-choice answers are shown below, in figure 5.2. The additional answers for the questionnaire have been added to the appendix.

	Strongly Disagree	Disagree	Neither Agree nor Disagree	Agree	Strongly Agree	Total
The instructions/guide provided was helpful and easy to understand.	0.00% 0	0.00% 0	0.00% 0	100.00% 3	0.00% 0	3
If Bingo is a suitable level of difficulty for GCSE students.	0.00% 0	0.00% 0	33.33% 1	66.67% 2	0.00% 0	3
If Bingo is engaging for students at GCSE level.	0.00% 0	0.00% 0	0.00% 0	66.67% 2	33.33% 1	3
I would use If Bingo again in the future to test students.	0.00% 0	0.00% 0	0.00% 0	66.67% 2	33.33% 1	3

Figure 5.6 – Questionnaire results

As a result of the comments received, the decision was made to address the confusion with the excel answer sheet. Providing the teacher with useful feedback was a key requirement, which was not achieved in the first version of the game. Therefor, the focus was on developing the teacher control panel before adding more levels of difficulty. A big advantage was the instant feedback teachers could provide to students.

Once completed, no feedback was received on the teacher control panel, despite having the website advertised for a month.

5.4 Testing

One of the requirements identified was for a suitable game length. Although the game is an innovative way of testing the students, it shouldn't take up too much of the lesson time. An ideal length is 10 minutes, with up to a minute per calling card.

To test the reliability of the game length (number of calling cards generated till a ticket has all squares completed), a test class was implemented. The aim of the class was to generate multiple instances of a bingo game and observe how many calling cards were generated before achieving bingo.

For all tests, a sample size of 1000 ticket sets was used. The first test contained 10 tickets per set. The output is shown in figure 5.3 below. The majority of games were completed in 7 to 10 calling cards, with an average of 8.2 calling cards till bingo.

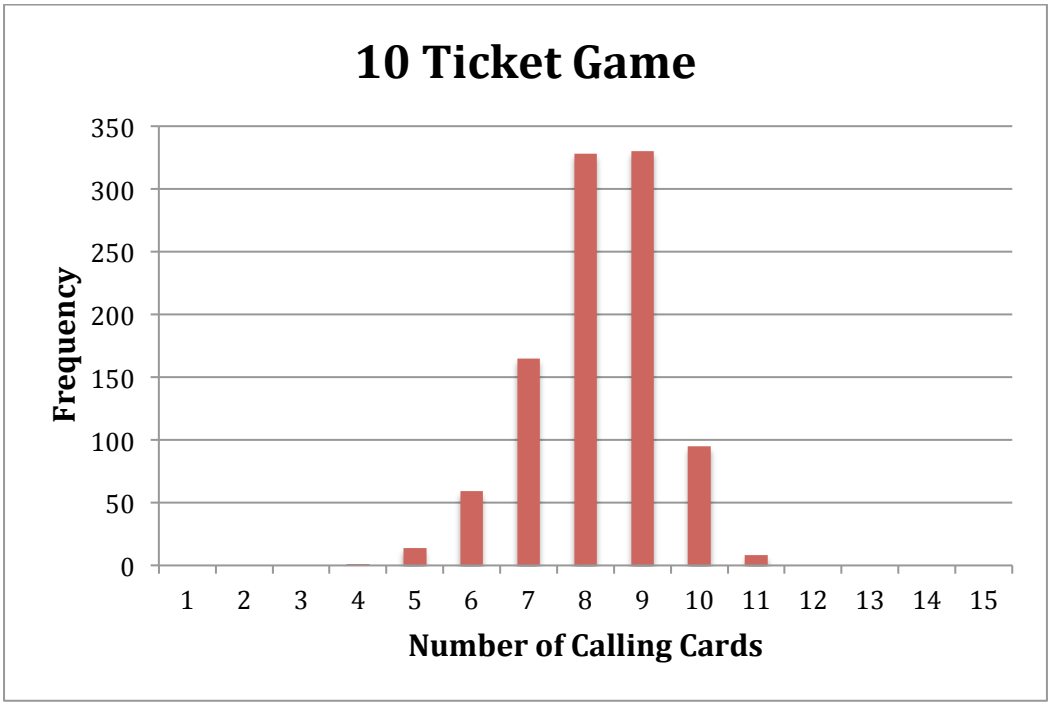


Figure 5.7 – Graph showing test results

The next set of tests was using sets containing 20 tickets. The average length of game is slightly higher at 8.7 calling cards till bingo, but still within an acceptable range. The output is shown in figure 5.4.

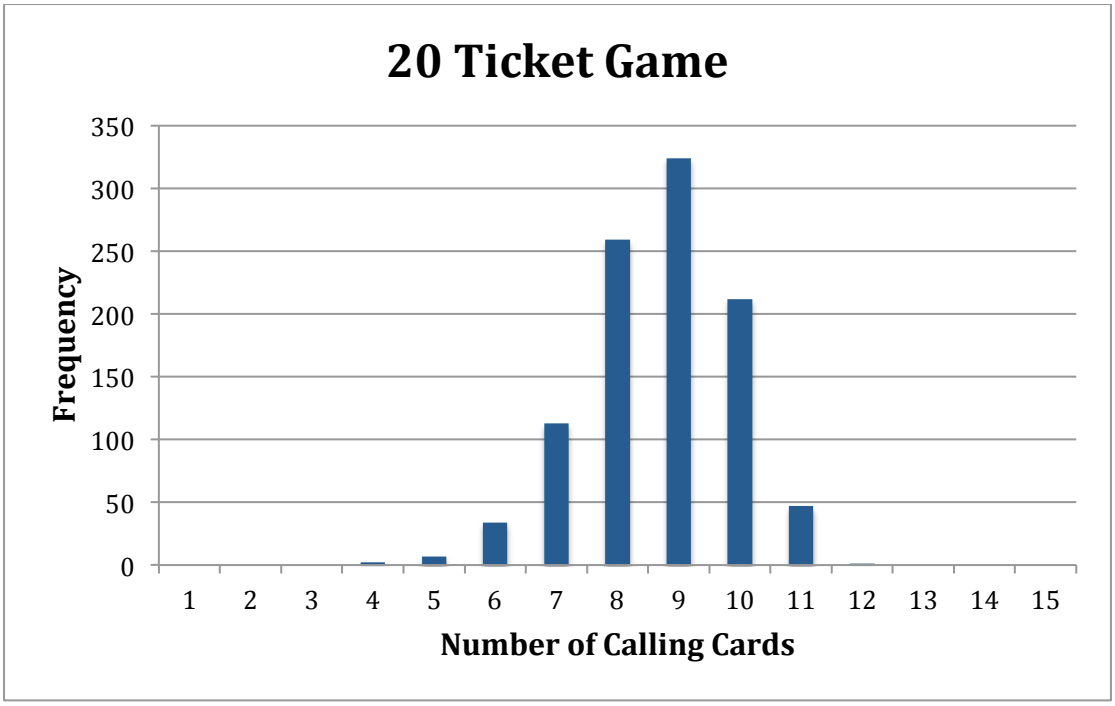


Figure 5.8 – Graph showing test results

The final test was for games with 30 tickets per set. Again, the average length of the game slightly increased to 8.9 calling cards till bingo. The output is shown in figure 5.5 below.

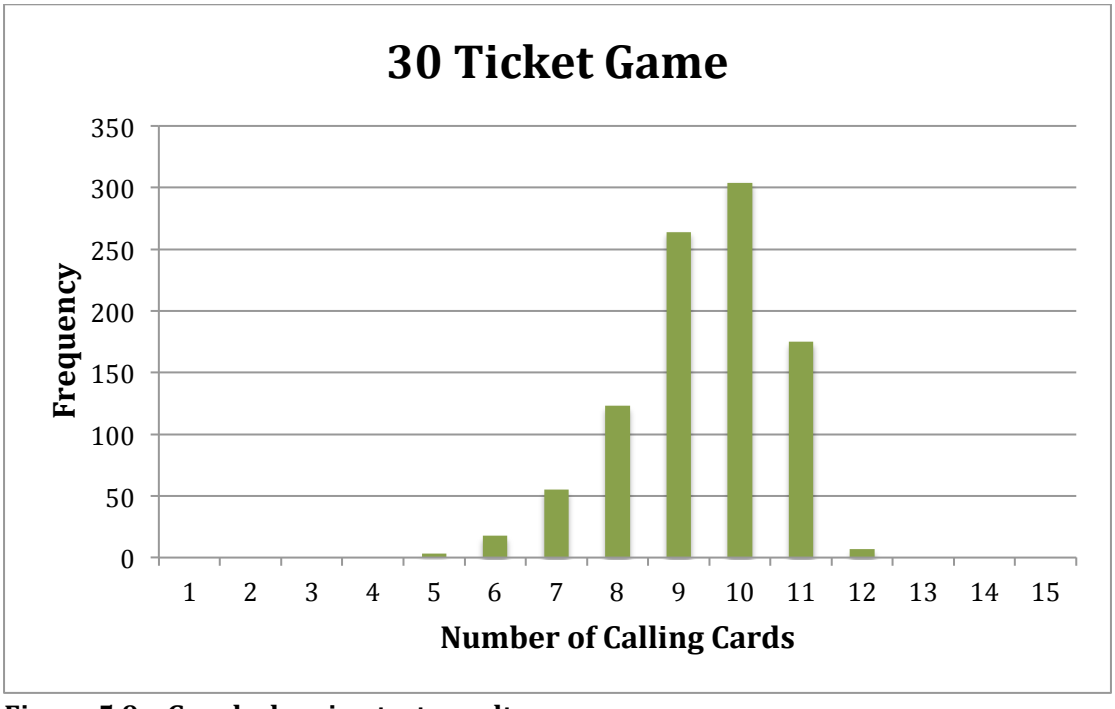


Figure 5.9 – Graph showing test results

The results from the tests are very encouraging and indicate the game length requirement has been adequately met.

6. Conclusion

After my initial research, it was clear there was a demand for additional resources for GCSE Computer Science courses. It's hard to learn a programming language by just reading guides, so teachers look for alternative methods of teaching/testing students. There is an almost constant influx of new resources on the CAS website, for teachers to download and give feedback if they wish.

It seems like the teachers preferred to use the first version of the game, rather than try the complete game. This may be because it is much simpler to use and doesn't require much work to set-up in the classroom. I think teachers would be more likely to use this if it was demonstrated to them, or possibly a YouTube video guide.
Nonetheless, the feedback received from teachers was very positive for the first version. After adding more features to the game, I hope teachers would find the final version easier to use and much more customizable.

6.1 Future Improvements

The great thing about this game is its adaptability and potential for growth. With more time, I could include more than just "if" statements. When deciding on which areas of the curriculum to focus on, it would be helpful to request the input of teachers. Specific topics may be more suited to the use of additional resources. Future additions could include:

6.1.1 Variable assignments

A topic that can often cause confusion when learning programming, but just requires a lot of practice. Like conditional statements, variable assignments have varying levels of complexity that could be defined by the teacher.
Below are examples of variable assignments, followed by a conditional statement. The conditional statement in both is basic, however this could also be altered to make it more challenging.

Basic	Advanced

`y = x + 2`
`if y < 2 :`
`  print("Bingo!")`
	`x = x*y`
	`y = x+7`
	`if y < 2 :`
	`  print("Bingo!")`

Figure 6.1 - Variable assignment examples

6.1.2 For loops

Program flow control is another topic stated on the GCSE specifications, which could be included as an option for the game. As there are multiple ways to integrate for loops in the squares, I will go through each with an example and short description.

The first example is using the range function, with a range of integer values. The most basic implementation of this would be to check if a variable is equal to any of the integer values in the range.

```
for a in range(1, 4):
   if y == a :
       print("Bingo!")
```

Figure 6.2 – Basic "for" loop example

A modification of this would be to use a "for" loop to alter the value of a variable. In the example below, the value of y is increased by 2 for each loop. Once the loop has completed, an "if" statement is entered.

```
for a in range(1, 4):
   y += 2
if y < 2 :
    print("Bingo!")
```

Figure 6.3 – "for" loop modifying the value of a variable

Alternatively, we could include the "if" statement within the "for" loop, checking if the condition is met after each iteration of the loop.

```
for a in range(1, 4):
   y += 2
   if y < 2 :
      print("Bingo!")
```

Figure 6.4 - "for" loop modifying the value of a variable with "if" statement in each loop

An advanced version of the previous examples would be to use nested loops. These would require more time for students to check each square, however it does ensure the students have a good level of understanding.

```
for a in range(1, 4):
   for b in range(1, 4):
      y = y + a + b
      if y > 12 :
        print("Bingo!")
```

Figure 6.5 – nested "for" loop

These examples are just some of the possibilities, which is very encouraging for the future of this game. I have thoroughly enjoyed developing the game, despite facing multiple challenges. The algorithms to generate tickets & calling cards required many hours of testing and modification, to meet the requirements of the game. To make sure the code within squares was relevant for the GCSE Computer Science, I had to spend a lot of time studying the specifications for multiple exam boards. The teachers who provided feedback on the game were a pleasure to work with and the positive feedback has certainly made the time and effort put into the project worthwhile.

7. References

AQA (2012) Specification [online] Available at: http://filestore.aqa.org.uk/subjects/AQA-GCSE-COMPSCI-W-SP.PDF [Accessed 3.12.2013]

Ceder V. and Yergler N. (2003) Teaching Programming with Python and Pygame Canterbury

Clarkson, M. (2012). The Unofficial Teacher's Guide [online]. Available at: http://www.ocr.org.uk/Images/139051-the-unofficial-teacher-s-guide-for-gcse-computing.pdf [Accessed 17.10.2013]

Computing at School [online] Available at: http://www.computingatschool.org.uk [Accessed 29.11.2013]

Gokhale, A. (1995) Collaborative Learning Enhances Critical Thinking [online] Available at: http://scholar.lib.vt.edu/ejournals/JTE/v7n1/gokhale.jte-v7n1.html [Accessed 20.10.2013]

Gove, M. (2011). Reforming qualifications and the curriculum to better prepare pupils for life after school [online] Available at: https://www.gov.uk/government/news/harmful-ict-curriculum-set-to-be-dropped-to-make-way-for-rigorous-computer-science [Accessed 21.10.2013]

Hazzan, O., Lapidot T. and Ragonis N. (2011). Guide to Teaching Computer Science: An Activity-Based Approach

OCR (2012). Specification J275 [online]. Available at: http://www.ocr.org.uk/Images/72936-specification.pdf [Accessed 17.10.2013]

Pearson Edexcel (2013). Specification [online]. Available at: http://www.edexcel.com/migrationdocuments/GCSE%20New%20GCSE/9781446908358_GCSE_Lin_CompScie_web.pdf [Accessed 17.10.2013]

Pearson Edexcel (2013). Scheme of Work [online]. Available at: http://www.edexcel.com/quals/gcse/gcse-2013/computer-science/Pages/default.aspx [Accessed 18.10.2013]

Royal Society (2011) Shut Down or restart? [online] Available at: http://royalsociety.org/education/policy/computing-in-schools/report/ [Accessed 18.10.2013]

Snowden R. (1986). Origins of Bingo [online] Available at: http://www.strangelife.com/bingodoc/bingohist.html [Accessed 01.12.2013]

Tickle L. (2012). How teachers and schools are preparing for the new computer science GCSE Available at: http://www.theguardian.com/teacher-network/2012/sep/13/computer-science-gcse-teachers-schools [Accessed 3.12.2013]

Vygotsky, L. (1978). Mind in society: The development of higher psychological processes

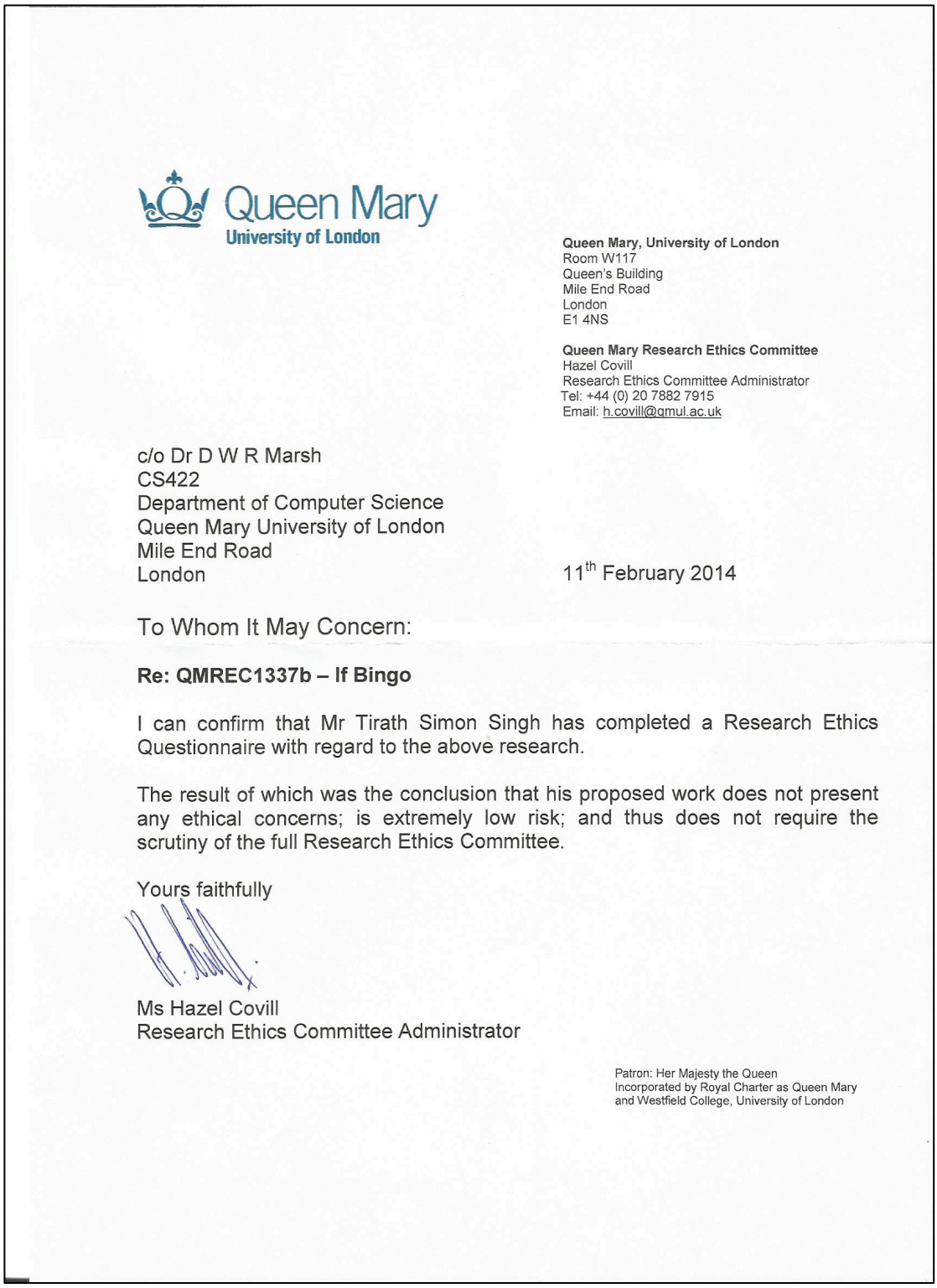
8. Appendix

Fast-track Ethics application

QMUL Ethics Filter – for research involving human participants

***Name**	**Tirath Singh**
Student Number if appropriate	**100130029**
***Email address**	**t.singh@se10.qmul.ac.uk**
Programme of Study (e.g. Undergraduate Geography Programme, Taught Education Masters)	**Undergraduate Computer Science**
***Department**	**Electric Engineering and Computer Science**
***Title of study**	**Computer Science with Industrial Experience**
***300 word minimum summary of the research:** This final year project is designing a resource for learning programming. I wish to get feedback from practicing teachers on the suitability of the resource for classroom use. The project has developed software for a game 'if bingo' that is used to test comprehension of some aspects of a programming language. The software generates sets of if bingo cards automatically, with the intention that these can be used by teachers to help test understanding of conditional statements in Python by pupils studying GCSE Computer Science in a fun way. The initial approach to teachers will be made by my supervisor, Dr Marsh, who will send a single message on behalf on a number of similar projects, asking for volunteer evaluators. The teachers contacted will be primarily those involved in the TLC (Teaching London Computing) project, in which Dr Marsh is a CI. Once a teacher declares interest in the game, I provide them with a PDF document containing bingo cards (generated by the program), a spreadsheet with answers to assist the teacher and a PowerPoint presentation. The presentation will provide basic rules for playing the game. The teacher will also be sent an online questionnaire, which is optional for them to complete and return. The questionnaire is voluntary and can be filled out anonymously and the names of participants will not be mentioned in the report. At most one reminder will be sent to complete the questionnaire.	
***Supervisor's (Principal Investigator) Name**	**William Marsh**
***Supervisor's email address**	**d.w.r.marsh@qmul.ac.uk**
***I confirm that Queen Mary University London is responsible for this study and that I am not receiving any funding for this project (other than that provided by myself or through my course)** **If in receipt of funding – who is the funding body** **Level of funding**	☐ **Yes** **N/A**

	Principle	Yes	No	Comments
1.	Are the participants under 16		**X**	Although the resources are intended for eventual classroom use, all our work is exclusively with teachers. We do not ask or require teachers to trial the materials in class, simply to use their professional judgement to give us feedback. Any decision about the classroom use of the resource rests entirely with teachers.
2.	Could the participants be classified as vulnerable adults		**X**	
3.	Do the participants have learning difficulties		**X**	
4.	Does the research involve using or collecting human tissue		**X**	
5.	Could this research uncover illegal activities (drug use, immigration etc.)		**X**	
6.	Could this research cause stress or anxiety in the participant		**X**	The teachers being contacted are ICT teachers who are learning programming. This process is demanding but voluntary participation in this evaluation does not add to the stress since teachers are free to drop out at any time.
7.	Will you be asking questions relating to issues of a personal sensitive nature		**X**	
8.	Could this research bring the University into disrepute		**X**	
9.	Does the research involve the person taking a drug of any description – even over the counter medicines		**X**	
10.	Does the research involve an intervention e.g. exercise, hypnotherapy		**X**	
11.	Does the research rely on covert observation of the participants		**X**	
12.	Will this research be conducted in the participants home		**X**	
13.	Will the participant be paid – not just expenses		**X**	
14.	Will the data collected be sent or used overseas		**X**	Anonymous feedback data will be used in the project report and may contribute to further publications. The data will not be used for evaluation in the TLC project.



Queen Mary
University of London

Queen Mary, University of London
Room W117
Queen's Building
Mile End Road
London
E1 4NS

Queen Mary Research Ethics Committee
Hazel Covill
Research Ethics Committee Administrator
Tel: +44 (0) 20 7882 7915
Email: h.covill@qmul.ac.uk

c/o Dr D W R Marsh
CS422
Department of Computer Science
Queen Mary University of London
Mile End Road
London

11th February 2014

To Whom It May Concern:

Re: QMREC1337b – If Bingo

I can confirm that Mr Tirath Simon Singh has completed a Research Ethics Questionnaire with regard to the above research.

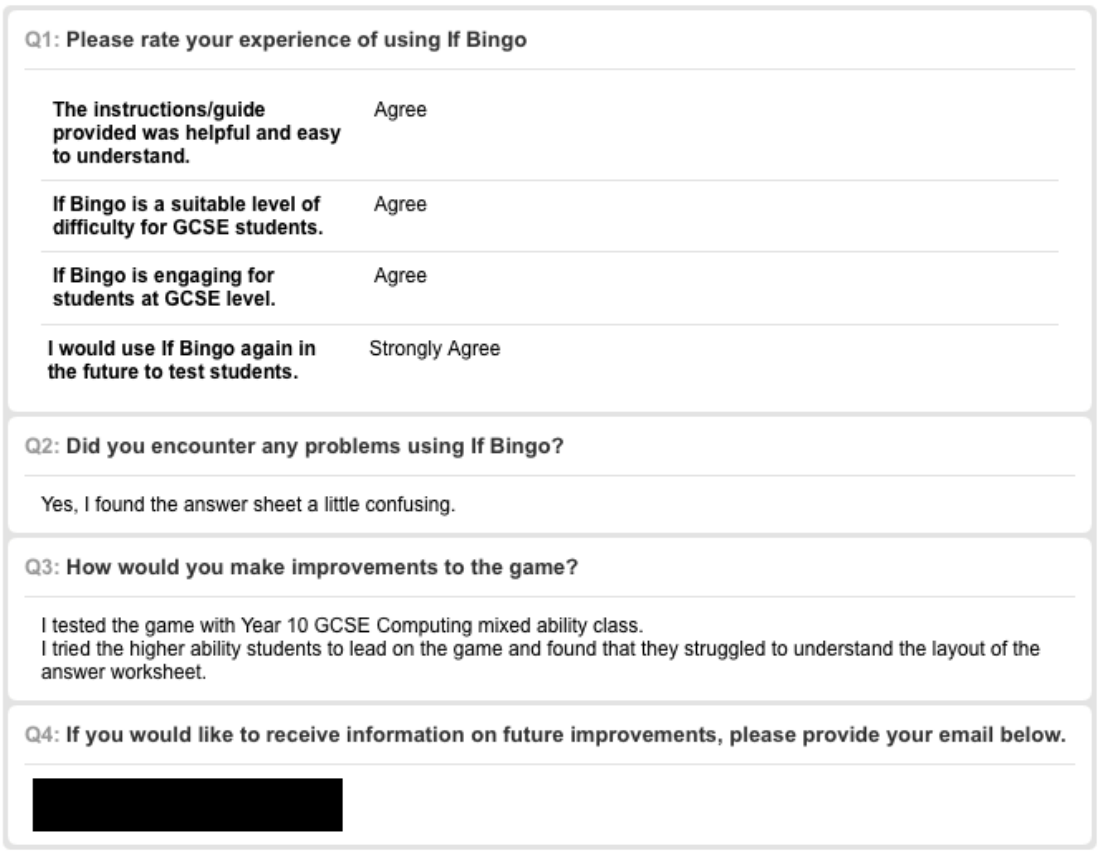
The result of which was the conclusion that his proposed work does not present any ethical concerns; is extremely low risk; and thus does not require the scrutiny of the full Research Ethics Committee.

Yours faithfully

Ms Hazel Covill
Research Ethics Committee Administrator

Patron: Her Majesty the Queen
Incorporated by Royal Charter as Queen Mary and Westfield College, University of London

Figure 8.1 - Research Ethics Approval letter



Q1: **Please rate your experience of using If Bingo**

The instructions/guide provided was helpful and easy to understand.	Agree
If Bingo is a suitable level of difficulty for GCSE students.	Agree
If Bingo is engaging for students at GCSE level.	Agree
I would use If Bingo again in the future to test students.	Strongly Agree

Q2: **Did you encounter any problems using If Bingo?**

Yes, I found the answer sheet a little confusing.

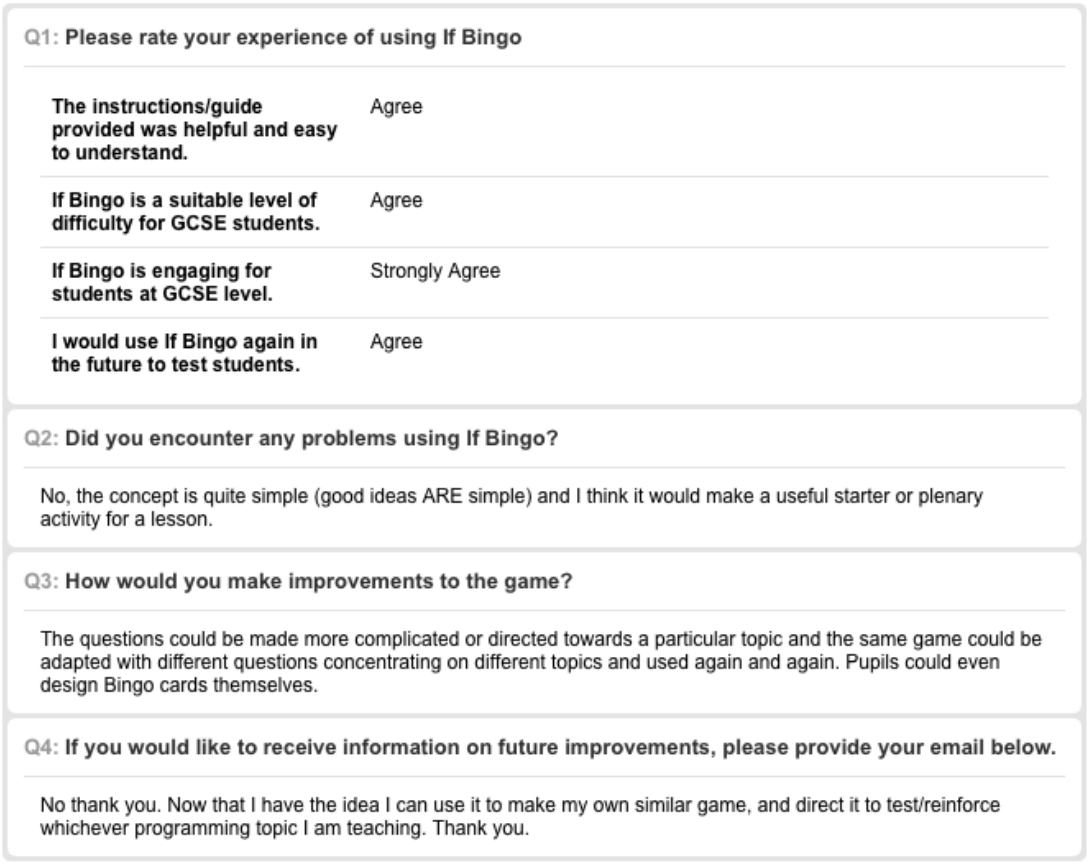
Q3: **How would you make improvements to the game?**

I tested the game with Year 10 GCSE Computing mixed ability class.
I tried the higher ability students to lead on the game and found that they struggled to understand the layout of the answer worksheet.

Q4: **If you would like to receive information on future improvements, please provide your email below.**

[redacted]

Figure 8.2 - Feedback A



Q1: **Please rate your experience of using If Bingo**

The instructions/guide provided was helpful and easy to understand.	Agree
If Bingo is a suitable level of difficulty for GCSE students.	Agree
If Bingo is engaging for students at GCSE level.	Strongly Agree
I would use If Bingo again in the future to test students.	Agree

Q2: **Did you encounter any problems using If Bingo?**

No, the concept is quite simple (good ideas ARE simple) and I think it would make a useful starter or plenary activity for a lesson.

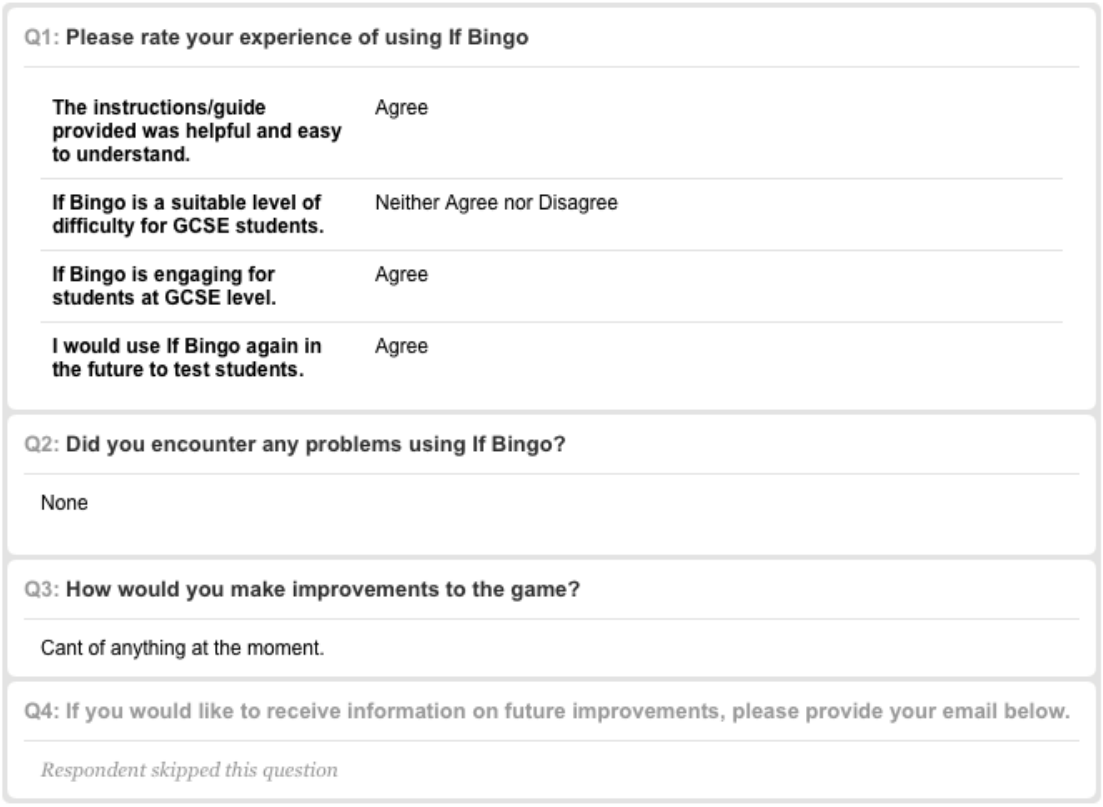
Q3: **How would you make improvements to the game?**

The questions could be made more complicated or directed towards a particular topic and the same game could be adapted with different questions concentrating on different topics and used again and again. Pupils could even design Bingo cards themselves.

Q4: **If you would like to receive information on future improvements, please provide your email below.**

No thank you. Now that I have the idea I can use it to make my own similar game, and direct it to test/reinforce whichever programming topic I am teaching. Thank you.

Figure 8.3 - Feedback B



Q1: **Please rate your experience of using If Bingo**

The instructions/guide provided was helpful and easy to understand.	Agree
If Bingo is a suitable level of difficulty for GCSE students.	Neither Agree nor Disagree
If Bingo is engaging for students at GCSE level.	Agree
I would use If Bingo again in the future to test students.	Agree

Q2: **Did you encounter any problems using If Bingo?**

None

Q3: **How would you make improvements to the game?**

Cant of anything at the moment.

Q4: **If you would like to receive information on future improvements, please provide your email below.**

Respondent skipped this question

Figure 8.4 - Feedback C

```
public class Test {
    public static void main(String[] Args){
        int[] sums = new int[15];
        for(int i=0; i<1000; i++){
            int tNum = 30;
            boolean advanced = false;
            boolean java =false;
            Generate g = new Generate(tNum,advanced,java);
            boolean generated = g.createTicketSet();
            while(!generated){
                g = new Generate(tNum,advanced,java);
                generated = g.createTicketSet();
            }

            boolean b = g.createCards();
            if(!b){
                System.out.println("Fail");
            }else{
                int tick = g.getTickBingo();
                sums[tick]++;
            }

        }
        for(int i=0; i<15;i++){
            System.out.println(sums[i]);
        }

    }
}
```

Figure 8.5 – Java class to produce test results