Maths Game

A game to teach the concept of Simultaneous Equations

Sorcha S. Nic Amhalaí

Final Year Project - 2005
Computer Science & Software Engineering

NUI MAYNOOTH
Ollscoil na hÉireann Má Nuad

Department of Computer Science
National University of Ireland, Maynooth
Co. Kildare
Ireland

A thesis submitted in partial fulfillment of the B.Sc degree in Computer Science and Software Engineering

Supervisor: Dr. Paul Gibson

Declaration

I hereby certify that this material, which I now submit for assessment on the program of study leading to the award of B.Sc. in Computer Science and Software Engineering, is entirely my own work and has not been taken from the work of others - save and to the extent that such work has been cited and acknowledged within the text of my work.

Signed: ___________________________________ Date: ___________________

Abstract

The aim of this project was to design and implement an educational game to teach a mathematical concept. The desired result is that secondary school students who have played the final game would perform better in questions involving this mathematical concept than students that have not.

The project work can be divided into two stages, the first stage taking place in the first semester of term and the second stage taking place in the second semester of term. The first stage consisted of analysing and improving the software development skills available. The second stage consisted of the gathering of requirements, design, implementation and testing of a functioning prototype of the final game. This report is primarily concerned with this stage.

Acknowledgments

I would like to thank the following people for their help in testing the game and their suggestions for improvements: Brian Nisbet, Anton Sweeney, Rebekah Burke, John "JC" Clarke, Dave Bolger, Jerry Cronin, Gareth Kavanagh, Gary Cosgrave and several others who declined the opportunity to be named.

I would also like to thank Paul Gibson for his hard work, support and direction. I would like to thank my father, Sean, my aunt, Sr. Pauline and my boyfriend's mother Debbie Wilson for their suggestions, insights and ideas into the world of teaching maths to children.

I would like to thank the Minds Computers Society. Minds provided secure backups, storage space and web space. The Society also gave a talk on thesis writing and introduced me to the wonders of LaTeX. As well as all this the members of the Society were always ready to give well informed and friendly technical advise. Thanks guys, you made this whole process a lot less painful.

I would also like to thank all my friends and family for their love and support.

Contents

Chapter 1

Introduction

1.1 Aims and Project Execution

The aim of this project was to design and implement an educational game to teach a mathematical concept. The original specification was to produce a game which would teach the concept of mathematical variables to children who were learning about them in school. The concept was discussed with secondary school teachers. These discussions indicated that secondary school students had real trouble understanding simultaneous equations. As a result the specification evolved to a game which would teach the concept of simultaneous equations. The children which would be using this game would be slightly older than the children that would be using a game based on the original idea of teaching about variables because the concept of simultaneous equations is covered later in the curriculum. The desired result is that secondary school students who have played the final game would perform better in questions involving simultaneous equations than students that have not. It was not within the scope of the project to test this premise.

The project work can be divided into two stages, the first stage taking place in the first semester of term and the second stage taking place in the second semester of term. The first stage consisted of analysing and improving the software development skills available. This was done primarily through the development of a prototype game. This prototype game was designed with the original concept of a game to teach the concept of variables in mind. The code for this prototype game and a screenshot of the game can be seen in Appendix C. The second stage consisted of the gathering of requirements, design, implementation and testing of a functioning prototype of the final game. This report is primarily concerned with this stage.

1.2 Process Development Model

The process development model used was a customer oriented rapid prototype development model. The customer, in the person of Dr. Paul Gibson, was consulted throughout the design and implementation phases. This meant that the game design could be easily adapted to suit the broad and evolving requirements of the customer.

1.3 Teaching Maths Through Games

Games are a tool which has long been used in the teaching of maths. However, until more recent times teaching was more usually done through game analysis rather than through actual game play. The idea of designing games specifically to teach mathematical concepts is quite a new one. This project is based on this idea. There are many other games available which are designed to teach maths. The primary differences between the game produced by this project and other games based on a similar idea that are available are that it is small, free and aimed at a very specific area. The fact that it is aimed at a specific problem area means that it should be easy to test to see if the game has had the desired effect.

1.4 Overview of the Report

This report is laid out in approximate chronological order: from requirements, through design, implementation and testing to future work and maintenance. However, due to the nature of the development model being used, many of these sections interact and many events which would traditionally occur later in the development process affected events which would traditionally occur earlier in the process. Where possible this is documented in the appropriate chapters.

The project, the process development chapter and the theory behind teaching maths through games are introduced in this chapter. Chapter 2 describes the requirements of the system. It investigates the reason for and advantages of having clearly articulated requirements for a system. The high level and low level requirements of the system are listed and the relationship between them is investigated. Chapter 3 deals with the design of the final system, including the design of the game, the design of the maze (which is an important part of the game) and the process of designing the system itself. It then goes on to describe in more detail the 'UML Light' notation used in the design process. Chapter 4 describes the implementation of the final product, including the technologies used, the evolution of the front end of the system and design decisions made during implementation. Chapter 5 describes the testing which has been done on the system to date including external black box testing and the continuous unit testing performed during implementation. Chapter 6 goes on to describe the future work and maintenance required on the system, including possible resolutions to issues discovered during testing, possible extensions to the game, other possible improvements to the game and maintenance.

Three appendices are included. Appendix A contains the code for the final game, including the code for the back end **backEnd.java**, the code for the front end **Game.java**, the html code for the main page (which includes the applet) **Game.html** and the html code for the rules page **Rules.html**. Appendix B contains the log produced during the second stage of the project and the implementation of the final system. Appendix C contains a screenshot of and the code for the original prototype game produced in the first stage of the project.

Chapter 2

Requirements

The requirements for this project can be separated into two broad categories, the 'high level' requirements and the 'low level' requirements. The high level requirements, which apply to any game that the project could have produced, did not change a lot from the beginning of the project. The low level requirements are specific to the final game that was produced and as such were defined only in the second stage of the project. That the low level requirements fulfill the high level requirements is investigated in the Requirements Analysis section below.

The low level requirements provided the correct level of abstraction. They allowed freedom to make design decisions such as how information would be taken from the user and how the various elements of the game would interact with each other. They also provided enough detail to give a clear idea of how the game should be implemented.

The requirements also provided a useful framework for testing. The tests that were developed from the requirements can be seen in section 5.2 of the Testing chapter along with their results.

2.1 High Level

The aim of the project was to design and implement an educational game to teach a mathematical concept. In order to achieve this the following high level requirements were specified.

The game is required to:

- be playable with mouse and / or keyboard
- be placed in a html page
- convey the mathematical concept being ‘taught’
- be expandable to allow recording of progress
- contain levels of difficulty
- be visual
- involve little or no typing
- contain some way to reset the game
- be possible to implement with time and skills available

2.2 Low Level

The low level requirements of the game were not specified formally. The following specifications were reached through conversation and discussion with the customer. A more detailed description of the process used to reach these requirements is given in Section 3.1 of the Design chapter.

The game is required to:

- be displayed in a html page
- get messages to user in some way
- take input from the user in some way
- contain a reset button which will reset the game when pressed
- contain a maze displayed on the screen with the following properties
    - the maze contains a Trolly, a Hole, a Checkout, an Exit and Bags
    - Bags are randomly generated with a number of apples and a number of bananas which are in some way visible to the user
    - when Bags are in the same square in the maze as the Hole they disappear

- when Bags are on the same square in the maze as the Trolly the Trolly will collect them
- the Trolly moves around the screen as the user presses the arrow keys
- when the Trolly collects a Bag the apples and bananas in the Bag are added to its own and the total price of all the apples and bananas in the Bag are added to its price
- the Trolly starts with the number of apples, the number of bananas and the price all set to zero
- the price for apples and bananas will be set randomly at the start of the game and will remain constant until the game has been won or reset
- when the Trolly lands on the Checkout it will display its contents to the user
- when the Trolly lands on the Exit it will ask the user for the price of one apple and the price of one banana and the player must get them right to win

2.3 Requirements Analysis

It is important that any game implemented as part of the project meet the high level requirements. Below each high level requirement is listed along with a comment on how the low level requirements meet it.

- **The game is required to be playable with mouse and / or keyboard:**
  This requirement is met by low level requirements because the game is played using the arrow and number keys.
- **The game is required to be placed in a html page:**
  This high level requirement is also listed as a low level requirement because the completion of this requirement is more of a matter for design and implementation than for requirements specifications.
- **The game is required to convey the mathematical concept being 'taught':**
  The low level requirements were devised with this requirement in mind. However, until the game has been implemented and tested in the field it is difficult confirm that this requirement has been met.

- **The game is required to be expandable to allow recording of progress:**
  Although there is nothing specific in the low level requirements that covers this high level requirement, the low level requirements were devised with this in mind. Some possible implementations of this requirement are discussed in section 6.2.2 of the Future Work chapter.
- **The game is required to contain levels of difficulty:**
  Similarly, this requirement is not directly covered by the low level requirements but is covered in section 6.2.1 of the Future Work chapter.
- **The game is required to be visual:**
  The low level meets this requirement because the game specified by the low level requirements is based on the player interacting with the maze and the objects in it by moving the Trolly.
- **The game is required to involve little or no typing:**
  It was decided that implementing the concept behind the game specified by the low level requirements with no typing was not feasible. The idea of implementing multiple choice answers was briefly discussed. It was dismissed because it was not felt that multiple choice answers would be as good an indication of the player's understanding of simultaneous equations.
- **The game is required to contain some way to reset the game:**
  The low level requirements include a reset button.
- **The game is required to be possible to implement with time and skills available**
  It was felt that the skills were available to produce a functional prototype of this game within the time available. The first stage of the project, which culminated in the production of the first prototype game (see Appendix C) was useful in determining what skills were available.

Chapter 3

Design

3.1 Game Design

The design of the game was done in conjunction with the customer and the result was the requirements outlined in Section 2.2 of the Requirements chapter. The intention was to design a game which fulfilled the high level requirements listed in Section 2.1 of the Requirements chapter. During the design of the game many ideas such as displaying the algebraic and Euclidian representations of the equations in question were suggested and dismissed. It was thought that the most simple implementation of the game would probably be the best, both from a game play point of view and an implementation point of view.

3.2 Maze Design

The implementation of the final game required the use of a maze. The concept of using random maze generation was examined and considered to be outside the scope of the project. The possibility of extending the game to contain multiple mazes, possibly randomly generated, is discussed in Section 6.3 of the Future Work and Maintenance chapter.

The maze used in the current version of the game was designed in Microsoft Excel [8].

After unit testing a number of alterations were made to the design. These changes were first made to the Excel design. The changes were then made to the hard coded maze in the code for the game itself.

3.3 Design Process

The back end was designed before the front end as it was felt to be more important to the production of a functioning prototype. Due to the need to keep a track of the classes, methods and variables that would exist within the code it was decided to produce specifications. A variation of UML known as UML Light was used, it is described in more detail in Section 3.4. The basic specifications for the back end varied only very slightly during the implementation. The changes to the back end design which did take place are discussed in more detail in the Implementation chapter. The final version of the back end specifications can be seen in Figure 3.1.

The original intention was to follow a similar design process for the front end as was followed for the back end. However, in practice, the front end evolved from the dummy front end which was used to test the functionality of the back end. The appearance of the game was evolving because it was secondary to the functionality. A more detailed description of the evolution of the front end can be seen in Section 4.2 of the Implementation chapter. A UML Light specification was kept. Although it changed greatly between the initial and final versions, this specification was very useful in implementing the front end. The final version can be seen in Figure 3.2.

3.4 UML 'Light'

A method of specification was required to keep track of what class, methods and variables that would exist. After investigation it was felt that full UML was not required. The specifications were drawn up in UML 'Light'. UML notation was used to keep a track of the classes, methods and variables. Other aspects of UML modeling were thought to

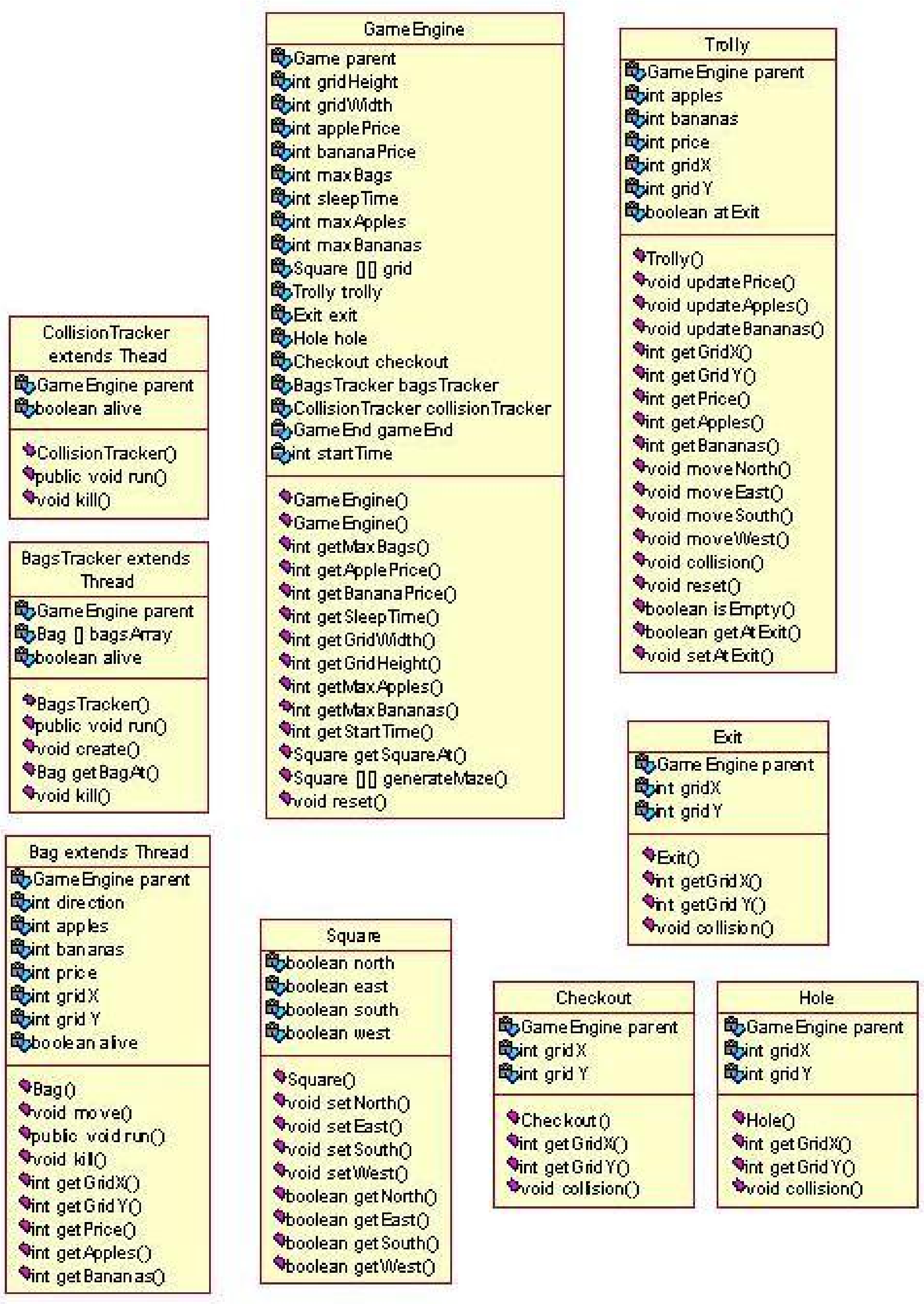


Figure 3.1: Back End UML ‘Light’

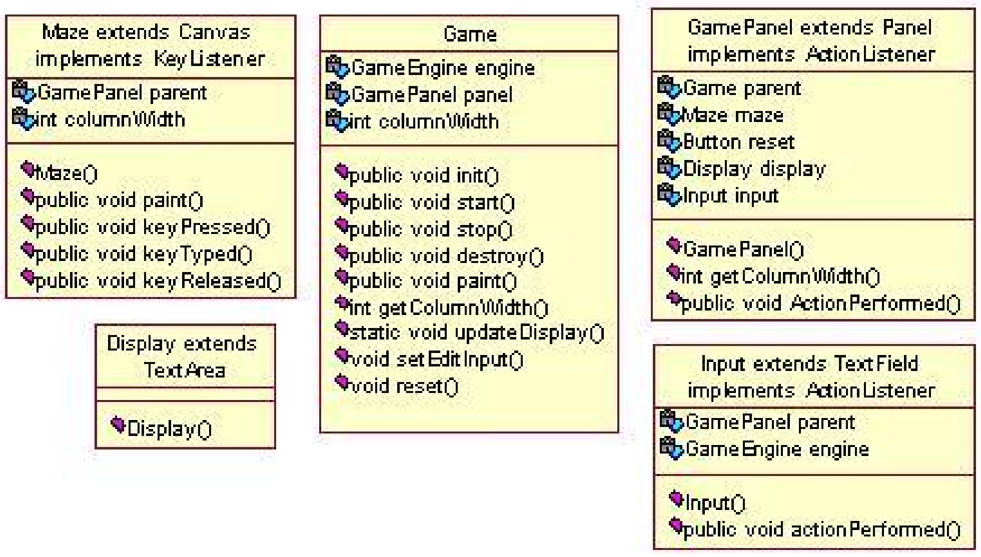


Figure 3.2: Front End UML 'Light'

be unnecessary. Examples of UML Light can be seen in Figures 3.1 and 3.2. Note that as well as the names and return types of methods, the number, type and name of the parameters to the methods were also recorded in the UML Light but were not displayed here. These specification were drawn up in Rational Rose [9]. This UML Light notation was found to be extremely useful during the project. This was particularly true during the implementation phase where it was used as a reference for such things as the return types of methods when deciding what methods to use in a particular situation. It also provided a useful level of abstraction, displaying similarities, differences and relationships between classes.

Chapter 4

Implementation

4.1 Technologies Used

4.1.1 Java

Java [6] was chosen as the implementation language because it was useful for the following requirements:

- **The game is required to be placed in a html page:** The functionality of **Applets** allowed the game to be placed in a html page.
- **The game is required to be possible to implement with the time and skills available:** Using Java provided all the functionality required without the necessity of learning a new language.

Put something in here about what version of Java is needed.

4.1.2 Tortoise CVS

It was decided that the project was large enough to require a version control system. The version control system chosen was Tortoise CVS. CVS (Concurrent Versions System) [1] is an open-source network-transparent version control system. Tortoise CVS is a Windows implementation of CVS which allows the user to interact with the CVS repository directly from Windows Explorer. Tortoise CVS is produced by Source Forge .net [2] and licensed under a GNU General Public License. The CVS repository was set up on the Minds Computer Society's server. Use of the Minds server assured a secure backup of the project. After the initial set up the use of the system was very straight forward.

Tortoise CVS was particularly useful for recording changes. Each time a new version was completed the new version would be uploaded, along with a comment on the changes. It also contains functionality to compare the current version with the previous version. This was very useful in determining exactly what changes had been made.

The system was not used to its full potential from the beginning. Earlier versions were sometimes entered too infrequently and sometimes entered with insufficient comments. This was recognised and remedied later in the project.

Version control is particularly useful when a change needs to be undone. This was used once during the project. A bug fix was implemented when it was discovered that the front end was not refreshing every time there was a change in the back end. It was later decided to resolve this problem in a different way. Tortoise CVS proved invaluable in regressing this change.

4.2 Evolution of the Front End

The design of the front end evolved during the implementation phase. The current front end evolved from the dummy front end intended to show that the back end functionality was operating correctly. Some of the changes made to the front end were made for purely visual reasons, others were made to reflect the structure of the back end or improve game play. Many other changes which were made were considered to be bugs and were fixed. These changes are discussed in Section 5.4 of the Testing chapter.

At first the Bags were represented on screen by the letter B, the Trolly was represented by the letter T and the Hole, Exit and Checkout were represented by the letters H, E and C respectively. This was done to test the functionality of the movements and locations of the objects.

The Trolly and Bags were then updated so that the number of apples contained in the Trolly or Bag was displayed in green and the number of bananas was displayed in yellow. This was done so that it would be possible to differentiate the number of apples from the number of bananas to be aware of the number of each contained in the object in question. The background colour of the maze was originally changed to make the numbers more visible. Later the background colour was changed back to its original colour and the numbers were displayed in black on green and yellow backgrounds instead of being displayed in green and yellow. Coloured boxes were added around the Trolly and the Bags to differentiate between them. The Hole, which had originally been represented by the letter H, was replaced by a black square. The Exit and Checkout, are, in the current version, still represented by the letters E and C respectively. The intention is that they will be replaced by suitable graphics. This is discussed in more detail in section 6.1 of the Future Work chapter.

Many different sizes of maze were implemented and tested before the current size was selected as the most suitable. The size selected was the largest size possible to where the entire game could be viewed within a web browser, assuming that the user had at least one toolbar open.

In order to provide the visual experience desired, the decision was made to use the **BoxLayout**. In order to do this is was necessary to include the swing library [7]. Although this required the user to have a later version of Java installed it was considered to be the correct decision because it allowed the desired visual effect to be produced.

4.3 Implementation Decisions

Some design decisions were made during the implementation phase of the project.

GameEnd: An example is the inclusion of the **GameEnd** class which was not part of the original back end design. During the implementation phase it became clear that a separate class was required to test if the player had won the game when they entered input. This was achieved by the inclusion of the GameEnd class in the back end.

Collisions: Another part of the back end which was not in the first draft of the design is the mechanics of collisions. The **Trolly**, **Hole**, **Exit** and **Checkout** classes each have a method called **collision**. There is also a class called **CollisionTracker** and an instance of this class in the **GameEngine** class, which is the main class of the back end. The CollisionTracker class extends (is a subclass of) **Thread** and while it is running it checks repeatedly to see if any of the objects on the screen are in the same location and calls the appropriate collision method if they are. How these collisions were implemented and tracked was a design decision but it was made during the implementation phase. A bug was noticed with this implementation and is discussed in the Testing chapter.

Repainting: The original requirements specification did not indicate when the front end needed to be repainted. In the first implementation tried the front end was repainted each time a key was pressed. This implementation was flawed however as the game needed to refresh each time a **Bag** moved as well as when a Trolly did. The first fix that was tried was to have a Thread at the front end which repainted the screen repeatedly. This was later replaced by a solution that involved repainting each time a **move** method was called within a Bag or the Trolly by calling a **repaint** method in the **Game** class (which is the main class in the front end)

Listeners: There were many **listeners** involved in the front end. In most cases the decision was made to implement the object itself as its own listener. However, in order to make the reset button listen for itself it would have been necessary to create a subclass of the **Button** class which is already included in Java and is otherwise perfectly suitable. As a result it was decided to implement the listener in **GamePanel** which contained the reset button.

4.4 Separating Back End and Front End

The original requirements of the game did not specify that the front end and back end should be separate. As the design developed it became clear that the back end and front end could be separated into distinct entities. The code for the back end and the code for the front end were kept in separate files, as were the specifications. It appeared that either the front end or the back end could be replaced and the interface retained. However, in order to make this possible a few small changes had to be made to reduce the visibility that the back end had into the front end and vice versa. For example the **updateDisplay** method was implemented to replace the code **parent.panel.display.appendtext** in the back end which allowed the back end too much visibility into the front end. Similarly the **setEditInput** method was added to replace the code **panel.input.setEditable**. The method **BagsTracker.getBacktAt(i)** method was put in to replace the calls to **BagsTracker.BagsArray**[i] in the front end which allowed the front end too much visibility into the back end.

The functionality that tested whether or not the player had won the game was originally in the front end, because this was where the input was taken in. This was moved to the back end because it made more logical sense to be there. This was implemented in the back end using the textbfGameEnd class described above.

Chapter 5

Testing

5.1 External Black Box Testing

The prototype game was posted on the Mikado Society [3] and Games Society [4] forums and on the Irish Gaming [5] forum. Feedback was invited from the users of these sites. The people who gave feedback had no access to the code and no idea of the design or implementation details of the game. The feedback received was very useful, providing many insights into the game. The comments below are grouped into sections according to the theme addressed in the comment. Some of the comments are in multiple sections because they dealt with multiple themes.

Comments on Playing and Winning

The comments in this section refer to the mechanic for winning the game. Most of them refer to the known problem that the fact that Bags with 0 apples or 0 bananas make the game very easy to win. The appearance of these Bags at early levels of the game only could be used as a level mechanic. This concept is discussed in more detail in Section 6.2.1 of the Future Work and Maintenance chapter.

- Comment: 'There should be an extreme version where like the price are given as cubic equations with only 1 real root. The rules could be shorter and it gets funky when there are more Bags on the screen, if you manage to get a 1:0 Bag you are on to a winner (maybe make sure none of these spawn?)

- Comment: second comment: not a bug as such, but something that you might want to change: in my second game, I got "1 apples plus 0 bananas costs 5". It was a bit easy to solve from there Wink
- Comment: ...I assume that the value of the Bags are generated randomly. i would suggest getting rid of 0 as an option as they provide a shortcut in the game, and make it very easy...
- Comment: you could add a scoreboard for people to have more competitive play (if you intend inflicting this on Actual young children...) - randomised mazes aren't too hard right?
- Comment: emm... too much 1/0 Bags - and 0/1 Bags... - lots of room for expansion to more variables...

Comments on Playability

Two main issues were raised in this area. The first is that the game may be difficult or impossible to play on a machine that does not have arrow keys. This is dealt with in more detail in the Future Work and Maintenance chapter. The other is a 'glitch' where the Trolly gets caught even though there is no wall in the direction it is attempting to move. The user was unable to replicate this issue, so it is assumed that it was caused by something unrelated.

- Comment: I got onto it using ie. Just finished the game. Rules are well explained I thought. There appears to a glitch. Sometimes it get's caught even if there is no wall in the direction it moves in.. It happened twice in the two times I played it
- Comment: Good game. - Suggestion ; - In rules, mention that Num Lock must be off, and that you control the trolley with the Num Lock keypad. - Took many button pressing to get that one.

Comments on The Look and Feel of the Game

The following comments are on the look and feel of the game. Due to the nature of the project, look and feel was secondary to functionality. Many of the suggested improvements below will be implemented in the next version of the game and are discussed further in the Future Work and Maintenance chapter.

- Comment: first comment is that the players trolley is too difficult to distinguish from the other trolleys in the store.
- Comment: It's working fine for me, 'tis fun Smile
- Comment: If you sit on the Hole Bags go into the Trolly instead of into the Hole
- Comment: ok some points. - you need to rework the rules of the game so that they are more user friendly for the child to learn. As a suggesting screen shots of what each object is. The rules made no sense until i saw the actual game. Also clicking the link for the rules should pop up a new window (small window like the PM window in Mikado, and here in the games forum) - Their should also be an explanation of how to move around, even if it is only an explnation that they first need to click inside the Maze box, then press the up arrow to move up and so on...
- Comment: 1)With the rules maybe a little more formating with some pictures of what looks like what as opposed to describing them.I think the length of the rules was a tad long.....Imho a good simple game should need the minimum explaination, it should almost be self explanitory.
- Comment: 2)Is there anyway of changing how the game refreshes? The constant blinking of the actaul maze is kinda annoying and may not sit well with someone with problems with flashing things....
- Comment: 3)About the problem with telling the different trolleys apart, how about just dropping the colours from the user's trolley to distingush it?
- Comment: 4)Maybe have colours under both the counter and the Exit to make it more obvious about where they are.
- Comment: 5)How about putting the text field to the right of the game as opposed to below it? It just look like a lot of wasted space to the right. Or you could just

put a version of the rules in the vacant space and save having to click to a different page

- Comment: 6)I missed the link back to the game on the rules page, anyway of just having it so you dont have to scroll down to get to the link.(Im using firefox btw).You have a link to the rules on the top of one page and the link back at the bottom of the next page.Imho it would be better if they were both on the top.
- Comment: Worked fine for me, tech wise. The Bags don't disappear fast enough ie there were too many of them after about 30 seconds of play in most cases, possibly making it a little too easy. - The other thing that got me was that the text box didn't scroll down, so I had to keep on manually scrolling to see amounts, the questions and the answers. Could it be made to auto-scroll?
- Comment: nice. - i've a 28 second record... - trollies in my local supermarket can hold more than 9 bananna's and apples. - (ok i havne't tested this theory... but i'm pretty sure... although coming up with a (semi) plausible reason for this not to be the case seems like my kind of challenge.)

Comments and Suggestions on Levels of Difficulty

The options below for levels of difficulty are discussed in the Future Work and Maintenance chapter.

- Comment: 'There should be an extreme version where like the price are given as cubic equations with only 1 real root. The rules could be shorter and it gets funky when there are more Bags on the screen, if you manage to get a 1:0 Bag you are on to a winner (maybe make sure none of these spawn?)
- Comment: suggestion: have the Bags go slower in the early levels and speed up in the later levels that will make it harder to catch the Bags you want to catch and make you run into the unwanted Bags
- Comment: as the level of the game increases (the number of different items) things get much harder.

5.2 Testing Based on Requirements

The following tests were performed to test if the game complied to the requirements. The tests are listed by the number of the requirement being tested followed by the date and time tested (or 'Ongoing' where no specific test was needed), the test and the result.

1 Ongoing The game is required to be displayed in a html page;

Page is available in html page at www.minds.may.ie/ saoili/Game.html

2 10/03/05 : 15:54 The game is required to get messages to user in some way: Test that messages are displayed on screen

Game prints in display box when user goes to Checkout with a non-empty Trolly, when user goes to Exit, when Trolly collects Bag which does not fit and when user guesses

3 10/03/05 : 15:57 The game is required to take input from the user in some way: Test that user can put input into text field

When user goes to Exit, user can input things int text field. Non-number inputs are ignored and number inputs are taken as guesses. Stops taking inputs after second incorrect guess

4 10/03/05 : 17:47 The game is required to contain a reset button which will reset the game when pressed:

Removes all Bags and starts again, moves Trolly to starting position and empties, does not clear display screen.

The game is required to contain a maze displayed on the screen with the following properties:

5 Ongoing The maze contains a Trolly, a Hole, a Checkout, an Exit and Bags:

Successful

6 Ongoing Bags are randomly generated with a number of apples and a number of bananas which are in some way visible to the user:

Successful

7 25/02/05 : 14:33 Watch Bags to check they fall down the Hole

Yes, they do, eventually. Happens rarely with current maze layout. This is possibly a good thing

8 25/02/05 : 14:34 Does Trolly collect Bags

Successful

9 Ongoing the Trolly moves around the screen as the user presses the arrow keys:

Successful

10 10/03/05 : 17:49 The Trolly starts with the number of apples, the number of bananas and the price all set to zero:

Difficult to check price but otherwise successful

12 25/02/05 : 14:36 Test that Trolly's apples and bananas always go up by amount on Bag:

Successful

13 10/03/05 : 17:51 The price for apples and bananas will be set at the start of the game (randomly) and will remain constant until the game has been won or reset:

Difficult to test but successful to the extent that is verifiable

14 Ongoing When the Trolly lands on the Checkout it will display its contents to the user:

Successful

15 Ongoing When the Trolly lands on the Exit it will ask the user for the price of one apple and the price of one banana and the player must get them right to win:

Successful

5.3 General Testing

The following tests were performed to test other aspects of the game. The tests are listed by the date and time tested (or 'Ongoing' where no specific test was needed), the test and the result.

25/02/05 : 14:35 What if Trolly is on Hole? - if no Bag there

No result

25/02/05 : 14:35 What if Trolly is on Hole? - if Bag there

Bag gets collected

25/02/05 : 14:24 Check Trolly can never go through walls

Successful to the extent verifiable. Also can always move if no wall

25/02/05 : 14:27 Watch to see if Bags ever go through walls

More difficult to test successful to the extent verifiable. Some walls Bags can't get to. Bags change direction at walls.

25/02/05 : 14:29 What happens when Bags get to Exit?

Seems to have no effect (this is a success)

25/02/05 : 14:30 What happens when Bags get to Checkout?

Cannot happen with current version of maze

25/02/05 : 14:37 What happens when it looses focus?

No effect on game but interaction no longer possible

25/02/05 : 14:38 What happens when Bags overlap?

One Bag gets printed over the other for under one second they occupy the same square

25/02/05 : 14:37 What happens when full Trolly 'collects' Bag

Prints 'That Bag won't fit in the Trolly' to the display and the Bag disappears.

25/02/05 : 14:40 What if Trolly 'collect's Bag that will overfill one side and not the other?

Prints 'That Bag won't fit in the Trolly' to the display and the Bag disappears.

25/02/05 : 14:41 What happens when game runs for long period?

When running with an appletviewer the game keeps playing but causes errors in command window

25/02/05 : 16:18 Still running from previous test.

Will play for a bit but goes too slowly to play and takes both inputs as one input at the Exit. This should be re-tested after flickering problem has been fixed

5.4 Continuous Testing During Implementation

The game was tested throughout the implementation process. Each functional version of the game was tested. Many bugs were noted and repaired during this phase.

Repainting: When the original design was implemented the repaint was only being called when a key was pressed. This bug was resolved during implementation and is discussed in more detail in the Implementation chapter.

Number of Bags Onscreen: The original implementation had up to 20 Bags onscreen simultaneously. When this implementation was tested it was clear that this figure was too high and the maximum number of Bags onscreen at once was reduced to 5.

Problem Generating Prices: During the ongoing testing it was noticed that the prices of apples and bananas, which were supposed to be randomly generated, were always being set to 0. The problem was resolved when it was discovered that the code which sets the prices was set to **(int)Math.random()*10+1** rather than **(int)(Math.random()*10)+1**. Math.random() returns a number between 0 and 1, when cast to an int it will always be 0, but when multiplied by 10 before casting to an int it will be a number between 0 and 9, which is the desired result.

Bag 'Spawn Spot': In the original design all the Bags would appear in the same location in the maze. This made it possibly for the player to sit on or near this 'Spawn Spot' and wait for the Bags to appear rather than having to collect them. This was a bug in game design rather than program design. It was easily fixed by making the Bags appear in whichever of two locations is further from the current location of the Trolly.

Collisions Being Called Repeatedly: As explained in the Implementation chapter, the functionality for **collisions** was added during the implementation phase. Once this functionality was implemented and tested a bug was noticed. When the Trolly was on either the Checkout or the Exit the appropriate collision method was called repeatedly. In the case of the collision between the Trolly and the Checkout this was quite easy to fix by emptying the Trolly within the collision method and only calling the collision method on a non-empty Trolly. In the case of the collision between the Trolly and the Exit the fix was more complicated. It involved creating a boolean value within the Trolly class which indicated whether or not the Trolly was at the Exit. This was more complicated to implement because the boolean needed to be set to false in all circumstances where the Trolly was no longer at the Exit, including where the game was won or reset and where the player moved the Trolly away.

Chapter 6

Future Work and Maintenance

The current version of the game is a fully functional prototype. There is a lot of scope for improvement, including those changes suggested during testing. This chapter deals with those possible improvements and possible extensions to the game.

Testing in the Field: The most important aspect of the future work is the field testing. The success or failure of the game to fulfill its intended purpose can not be determined until the game and its results have been tested in the classroom situation it was intended for. These field tests will not take place until the issues discovered during testing are resolved. More detail on these issues is given below. It would be preferable to have also implemented some of the concepts for extending the game before the field tests take place. These concepts are also detailed below.

6.1 Resolving Issues Discovered During Testing

The following issues were noted during the testing phase. They will, where possible, be resolved before the game is tested in the field.

Focus: The game will be updated so that when the player goes to the Exit the focus will go to the input box automatically. When they make their second guess the focus will go back to the maze automatically. The implementation of this concept will mean that the player will be able to play the game using the keyboard and remove the necessity for the use of the mouse.

Exit and Checkout: Currently the Exit and Checkout are represented by an E and a C respectively. The game will be updated so that the Exit and Checkout are represented either by suitable graphics or by squares of different colours, similar to the black square currently representing the Hole.

Distinguishing the Trolly from Bags: It was suggested in the comments received during external testing that it may be too difficult to tell the difference between the Trolly and the Bags. The coloured lines around the Trolly and the Bags will be made thicker in order to resolve this problem.

Flickering: This is a major problem with the current version of the game. The problem is that the game flickers on the screen and this problem gets worse when the game is left open for a long period of time. This problem is caused by the fact that the calls to repaint the game are occurring too frequently. One possible solution to this problem is to design the repaint so that only the parts of the game which have changed will be repainted. This solution would require a lot of work. The other option would be to repaint the front end only when all of the Bags have moved. An attempt was made at implementing this but was hindered by the fact that the number of Bags was constantly changing. One of these solutions or another solution to this problem will need to be implemented before the final version can be tested in the field.

Options for Movement: The current version of the game allows the player to move the Trolly only with the arrow keys. In the short term another set of keys for movement, probably i, j, l and m, will be implemented. This will be done to facilitate play on computers that do not have arrow keys. In the longer term mouse controlled movement may be implemented. This will require a great deal of research into what movement would be expected from the mouse. It will also be necessary to ensure that no advantage is gained by using the mouse over using the keyboard or vice versa.

Scrolling in Text Box: A minor problem reported during testing is that on some machines the text box does not automatically scroll to the bottom when new text is added to the display. This problem has been replicated on a number of machines but does not occur on all machines. More investigation is needed into this problem. However, resolving this problem is not a priority.

Revised Rules: The main feedback received on the rules was that they are two long and that they are too hard to follow without being able to see the game. It was also suggested that the rules should be displayed beside the game on the same web page. This idea will be implemented and should resolve the problem of the rules being difficult to follow without being able to see the game. The rules will also be rewritten shorter.

6.2 Extensions to the Game

6.2.1 Levels of Difficulty

The high level requirements include the requirement that the game contain levels of difficultly. It was not possible to integrate levels of difficulty into the game in the time allotted. The following are multiple ways of implementing levels of difficulty. Some of these will be implemented before the game is tested in the field.

Contents of Bags: The problem of Bags which had either 0 apples or 0 bananas was raised multiple times in the testing phase. It may be possible to use this as a game mechanic. In the first level of the game, some Bags would appear with either 0 apples and 1 banana or 1 apple and 0 bananas. In the second level some Bags would appear with either 0 apples and multiple bananas or multiple apples and 0 bananas. In the third level some Bags would appear with 1 apple and multiple bananas or multiple apples and 1 banana. In higher levels only Bags with multiple apples and multiple bananas would appear.

Speed of Bags: Levels of difficultly could be implemented by making the Bags move faster at higher levels. While this would increase the level of difficultly with the levels of play, it would not indicate an increase in understanding of simultaneous equations and as such will probably not be implemented.

More complicated maths: A suggestion was made from one of the external testers to include more complicated maths in higher levels. This would be difficult to implement and would probably require the game itself to be redesigned as well as the program. It also goes against one of the primary concepts behind the project in that it does not confine the game to one simple mathematical idea and will probably not be implemented.

Time Limit: Another possible technique for implementing levels of difficulty would be to introduce a time limit. This would require very little implementation work as the code already records the time taken to complete the game. It would however require some design work as the game would need to be redesigned to contain a visual display of the timer. This concept would be very compatible with the 'Contents of Bags' concept.

6.2.2 Measuring Player Progress

It is a high level requirement that it be possible to add functionality to measure the progress of the players. If the 'Contents of the Bags' and 'Time Limit' concepts for levels of difficulty then the level the players have reached will be a good indicator of their progress. Other methods for measuring progress include:

Profiles: The game could be altered so that it includes a log in system and records a player's progress along with their log in details. This would be difficult and expensive to implement.

Written Exams: The progress of the players could be tested by written exams based on simultaneous equations taken before and after playing the game. These would probably be a better indication of the progress of the players and would also provide a framework for the analysis of the success of the game.

6.3 Other Improvements

Other possible improvements to the game include:

Soft Coded Maze Size: The **columnWidth** variable in the **Game** class was originally included so that the size of the maze, including the elements within it, could be altered. During the implementation of the front end certain elements were hard coded. The result was that altering the value of the columnWidth variable no longer altered the size of the maze. These hard coded elements will be corrected so that the columnWidth variable can be used to alter the size of the maze.

Listening for Arrow Keys: During the implementation phase of the project it was decided to implement the listener for the key presses that would move the Trolly as part of the class that represents the maze. If the entire game were to implement the listener for these key presses then the maze would not need to have focus for the Trolly to move. This will be implemented and tested.

Maze Generation: The current version of the game contains only one maze which is hard coded. Future versions of the game will contain functionality to vary the layout of the maze. This may be achieved by random maze generation, reading mazes in from a file or by the game containing multiple stored mazes which may be randomly chosen or associated with specific levels.

6.4 Maintenance

The current version of the game will require very little maintenance. Implementing the changes proposed in the section above on 'Resolving Issues Discovered During Testing' should not affect the level of maintenance required. Some of the changes proposed in 'Extensions' and 'Other Possible Improvements' sections above could greatly increase the amount of maintenance required.

Chapter 7

Conclusions

7.1 Project Aims

The aim of this project was to design and implement an educational game that will teach a mathematical concept. As mentioned in the Future Work and Maintenance chapter, the success or failure of the project in this regard cannot be determined until field testing is completed.

7.2 Skill Learned and Knowledge Gained

During the implementation of this project I learned many valuable skills and lessons. My experience with CVS Tortoise taught me the importance of Version Control and the assistance it can provide in the production of a software product. I will use Version Control in future software projects and will take full advantage of the system with regular versions and meaningful comments.

I also learned the importance of design. The code associated with the back end of the final prototype is more cleanly and clearly implemented than the code associated with the front end. I believe that this is because more work was done on the design of the back end than on the front end. As a result I will put more work into the design of future software projects.

Appendix A

Code

A.1 backEnd.java

```
import java.util.*;

class GameEngine {
  Game parent;
  int gridHeight, gridWidth;
  int applePrice, bananaPrice;
  int maxBags, sleepTime;
  int maxApples, maxBananas;
  Square [][] grid;
  Trolly Trolly;
  Exit Exit;
  Hole Hole;
  Checkout Checkout;
  BagsTracker BagsTracker;
  CollisionTracker collisionTracker;
  GameEnd gameEnd;
  int startTime;

  GameEngine(Game p){
    parent = p;
    gridHeight = 11; /*hard coded for graphics*/
    gridWidth = 11; /*hard coded for graphics*/
    applePrice = (int)(Math.random()*10)+1;
    bananaPrice = (int)(Math.random()*10)+1;
    maxBags = 5; /*hard coded for now*/
    sleepTime = 20; /*hard coded for now*/
    maxApples = 9;
    maxBananas = 9;
    grid = generateMaze();
    Trolly = new Trolly (this);
    Exit = new Exit (this);
    Hole = new Hole (this);
    Checkout = new Checkout (this);
    BagsTracker = new BagsTracker (this);
    BagsTracker.start();
    collisionTracker = new CollisionTracker (this);
    collisionTracker.start();
    startTime = (int)(new Date().getTime());
  }//END constructor

  int getMaxBags (){
    return maxBags;
  }//END getMaxBags method
  int getApplePrice (){
    return applePrice;
  }//END getApplePrice method
  int getBananaPrice (){
    return bananaPrice;
  }//END getBananaPrice method
  int getSleepTime (){
    return sleepTime;
  }//END getSleepTime method
  int getGridWidth (){
    return gridWidth;
  }//END getGridWidth method
  int getGridHeight (){
    return gridHeight;
  }//END getridHeight method
  int getMaxApples (){
    return maxApples;
  }//END getMaxBags method
  int getMaxBananas (){
    return maxBananas;
  }//END getMaxBags method
  int getStartTime (){
    return startTime;
  }//END getStartTime method

  Square getSquareAt (int x, int y){
    return grid[x][y];
  }

  Square[][] generateMaze(){ /*hard coded for graphics*/
    Square [][] retVal = new Square [11][11];
    for (int i=0; i<11; i++){
      for (int j=0; j<11; j++){
        retVal [i][j] = new Square();
      }
    }
    for (int i=0; i<11; i++){
      retVal [0][i].setWest(true);
    }
    for (int i=0; i<11; i++){
      retVal [10][i].setEast(true);
    }
    for (int i=0; i<11; i++){
      retVal [i][0].setSouth(true);
    }
    for (int i=0; i<11; i++){
      retVal [i][10].setNorth(true);
    }
    retVal[4][10].setEast(true);
    retVal[2][9].setEast(true);
    retVal[3][9].setEast(true);
    retVal[4][9].setEast(true);
    retVal[5][9].setEast(true);
    retVal[1][8].setEast(true);
    retVal[3][8].setEast(true);
    retVal[4][8].setEast(true);
    retVal[7][8].setEast(true);
    retVal[8][8].setEast(true);
    retVal[2][7].setEast(true);
    retVal[3][7].setEast(true);
    retVal[6][7].setEast(true);
    retVal[7][7].setEast(true);
    retVal[0][6].setEast(true);
    retVal[9][6].setEast(true);
    retVal[0][5].setEast(true);
    retVal[9][5].setEast(true);
    retVal[0][4].setEast(true);
    retVal[9][4].setEast(true);
    retVal[2][3].setEast(true);
    retVal[3][3].setEast(true);
    retVal[4][3].setEast(true);
    retVal[2][2].setEast(true);
    retVal[2][2].setEast(true);
    retVal[3][2].setEast(true);
    retVal[4][2].setEast(true);
    retVal[6][2].setEast(true);
    retVal[0][1].setEast(true);
    retVal[1][1].setEast(true);
    retVal[3][1].setEast(true);
    retVal[7][1].setEast(true);
    retVal[1][0].setEast(true);
    retVal[8][0].setEast(true);
    for (int i=1; i<11; i++){
      for (int j=0; j<11; j++){
        if (retVal[i-1][j].getEast()){
          retVal[i][j].setWest(true);
        }
      }
    }
    retVal[1][8].setSouth(true);
    retVal[1][7].setSouth(true);
    retVal[1][3].setSouth(true);
    retVal[2][6].setSouth(true);
    retVal[2][5].setSouth(true);
    retVal[2][2].setSouth(true);
    retVal[3][6].setSouth(true);
    retVal[3][5].setSouth(true);
    retVal[3][1].setSouth(true);
    retVal[5][8].setSouth(true);
    retVal[5][2].setSouth(true);
    retVal[5][1].setSouth(true);
    retVal[6][10].setSouth(true);
    retVal[6][7].setSouth(true);
    retVal[6][4].setSouth(true);
    retVal[6][3].setSouth(true);
    retVal[6][1].setSouth(true);
    retVal[7][10].setSouth(true);
    retVal[7][9].setSouth(true);
    retVal[7][6].setSouth(true);
    retVal[7][5].setSouth(true);
    retVal[7][4].setSouth(true);
    retVal[7][2].setSouth(true);
    retVal[8][10].setSouth(true);
    retVal[8][6].setSouth(true);
    retVal[8][5].setSouth(true);
    retVal[8][3].setSouth(true);
    retVal[9][10].setSouth(true);
    retVal[9][9].setSouth(true);
    retVal[9][7].setSouth(true);
    retVal[9][4].setSouth(true);
    retVal[10][2].setSouth(true);
    for (int i=0; i<11; i++){
      for (int j=0; j<10; j++){
        if (retVal[i][j+1].getSouth()){
          retVal[i][j].setNorth(true);
        }
      }
    }
    return retVal;
  }//END generateMaze method

  void reset(){
    applePrice = (int)(Math.random()*10)+1;
    bananaPrice = (int)(Math.random()*10)+1;
    Trolly = new Trolly (this);
    BagsTracker.kill();
    BagsTracker = new BagsTracker(this);
    BagsTracker.start();
    collisionTracker.kill();
    collisionTracker = new CollisionTracker(this);
    collisionTracker.start();
    startTime = (int)(new Date().getTime());
  }//END reset method
}//END Class GameEngine
```

```
class Trolly {
  GameEngine parent;
  int apples, bananas, price, gridX, gridY;
  boolean atExit;

  Trolly (GameEngine p){
    parent = p;
    apples = 0;
    bananas = 0;
    price = 0;
    gridX = 0;
    gridY = 0;
    atExit = false;
  }//END constructor

  void updatePrice (int addedPrice){
    price += addedPrice;
  }//END updatePrice method
  void updateApples (int addedApples){
    apples += addedApples;
  }//END updateApples method
  void updateBananas (int addedBananas){
    bananas += addedBananas;
  }//END updateBananas method

  int getGridX (){
    return gridX;
  }//END getGridX method
  int getGridY (){
    return gridY;
  }//END getGridY method
  int getPrice (){
    return price;
  }//END getPrice method
  int getApples (){
    return apples;
  }//END getApples method
  int getBananas (){
    return bananas;
  }//END getBananas method
```

```
void moveNorth(){
  if (!(parent.getSquareAt(gridX,gridY).getNorth())){
    gridY++;
    parent.parent.repaint();
  }
}//END moveNorth method
void moveSouth(){
  if (!(parent.getSquareAt(gridX,gridY).getSouth())){
    gridY--;
    /*if atExit is true, we're moving away,
    so make it false*/
    if (atExit){
      atExit = false;
    }
    parent.parent.repaint();
  }
}//END moveSouth method
void moveEast(){
  if (!(parent.getSquareAt(gridX,gridY).getEast())){
    gridX++;
    /*if atExit is true, we're moving away,
    so make it false*/
    if (atExit){
      atExit = false;
    }
    parent.parent.repaint();
  }
}//END moveEast method
void moveWest(){
  if (!(parent.getSquareAt(gridX,gridY).getWest())){
    gridX--;
    parent.parent.repaint();
  }
}//END moveWest method

void collision (Bag b){
  if ((b.getApples()+apples>=parent.getMaxApples())
       ||(b.getBananas()
       +bananas>=parent.getMaxBananas())){
    parent.parent.updateDisplay
      ("That Bag won't fit in the Trolly\n");
      return;
    }
    updatePrice(b.getApples()*parent.getApplePrice());
    updateApples(b.getApples());
    updatePrice(b.getBananas()*parent.getBananaPrice());
    updateBananas(b.getBananas());
    b.kill();
    parent.parent.repaint();
  }//END collision method

  void reset (){
    apples = 0;
    bananas = 0;
    price = 0;
  }//END reset method

  boolean isEmpty (){
    return (price == 0);
  }//END isEmpty method

  boolean getAtExit(){
    return atExit;
  }//END getAtExit method

  void setAtExit (boolean b){
    atExit = b;
  }//END setAtExit method
}//END Class Trolly

class Checkout {
  int gridX, gridY;
  GameEngine parent;

  Checkout (GameEngine p){
    parent = p;
    gridX = parent.getGridWidth()-1;
    gridY = 0;
  }//END constructor

  int getGridX (){
    return gridX;
  }//END getGridX method
  int getGridY (){
    return gridY;
  }//END getGridY method

  void collision (Trolly t){
    parent.parent.updateDisplay
     (t.getApples()+" apples plus "+t.getBananas()
     +" bananas costs "+t.getPrice()+"\n");
    parent.Trolly.reset();
    parent.parent.repaint();
  }//END collision method
}//END class Checkout

class Exit {
  int gridX, gridY;
  GameEngine parent;

  Exit (GameEngine p){
    parent = p;
    gridX = 0;
    gridY = parent.getGridHeight() - 1;
  }//END constructor

  int getGridX (){
    return gridX;
  }//END getGridX method
  int getGridY (){
    return gridY;
  }//END getGridY method

  void collision (Trolly t){
    parent.gameEnd = new GameEnd(parent);
    parent.gameEnd.start();
    parent.parent.repaint();
  }//END collision method
}//END class Exit

class Hole {
  int gridX, gridY;
  GameEngine parent;
  Hole (GameEngine p){
    parent = p;
    gridX = ((int)parent.getGridWidth()/2);
    gridY = ((int)parent.getGridHeight()/2);
  }//END constructor

  int getGridX (){
    return gridX;
  }//END getGridX method
  int getGridY (){
    return gridY;
  }//END getGridY method

  void collision (Bag b){
    b.kill();
    parent.parent.repaint();
  }//END collision method
}//END class Hole

class GameEnd extends Thread {
  GameEngine parent;
  int guessed;
  boolean right;

  GameEnd (GameEngine p){
    parent = p;
    guessed = 0;
    right = true;
  }//END constructor

  public void run (){
    parent.parent.setEditInput(true);
    parent.parent.updateDisplay
     ("how much does one apple cost?\n");
    while ((guessed == 0)&&
     (parent.Trolly.getGridX()
     == parent.Exit.getGridX())
     &&(parent.Trolly.getGridY()
     == parent.Exit.getGridY())){}
    if (guessed == 1){
      parent.parent.updateDisplay
      ("how much does one banana cost?\n");
    }
    while ((guessed < 2)&&(parent.Trolly.getGridX()
           == parent.Exit.getGridX())
           &&(parent.Trolly.getGridY()
           == parent.Exit.getGridY())){}
    if (right&&(guessed==2)){
      parent.parent.updateDisplay
       ("Congratulations!\n");
      parent.parent.updateDisplay
       ("You completed the game in "
       + (((int)(new Date().getTime())
         -parent.getStartTime())/1000)
       +" seconds\n");
      parent.reset();
    }
    else {
      if (guessed==2) parent.parent.updateDisplay
      ("Try again\n"
      +"Please leave the Exit area and come back in\n");
    }
    /*moved out of else
    because reseting parent doesn't change it*/
    parent.parent.setEditInput(false);
  }//END run method

  void check (int i){
    if (((guessed==0)&&(i==parent.getApplePrice()))
        ||(guessed==1)&&(i==parent.getBananaPrice())){
      parent.parent.updateDisplay ("correct\n");
    }
    else {
      parent.parent.updateDisplay ("incorrect\n");
      right = false;
    }
    guessed++;
  }//END check method
}//END class GameEnd

class CollisionTracker extends Thread {
```

```
GameEngine parent;
boolean alive;

CollisionTracker (GameEngine p){
  parent = p;
  alive = true;
}//END constructor

void kill (){
  alive = false;
}//END kill method

public void run (){
  Trolly t = parent.Trolly;
  Checkout c = parent.Checkout;
  Exit e = parent.Exit;
  Hole h = parent.Hole;
  BagsTracker bt = parent.BagsTracker;
  Bag tempBag = null;
  int maxBags = parent.getMaxBags();
  /*previous line done here so only once*/
  while (alive){
    /*if the Trolly and the Checkout have the same
    co-ordinates, call the collision method of the
    Checkout with the Trolly as a paramater*/
    if ((t.getGridX()==c.getGridX())
        &&(t.getGridY()==c.getGridY())
        &&(!t.isEmpty())){
      c.collision(t);
    }
    /*if the Trolly and the Exit have the same
    co-ordinates, call the collision method of the
    Exit with the Trolly as a paramater*/
    if ((t.getGridX()==e.getGridX())
        &&(t.getGridY()==e.getGridY())
        &&(!t.getAtExit())){
      t.setAtExit(true);
      e.collision(t);
    }
    /*loop through all the Bags in the BagsTracker*/
    for (int i=0; i<maxBags; i++){
        tempBag = bt.getBagAt(i);
        if (tempBag != null){
          /*if the Bag is in the same place
          as the Trolly, call the Trolly's
          collision with the Bag as a parameter*/
          if ((tempBag.getGridX()==t.getGridX())
            &&(tempBag.getGridY()==t.getGridY())){
            t.collision(tempBag);
            bt.BagsArray[i]=null;
          }
        }
        /*needs to be in seperate if to check for null
        because previous if could set it to null*/
        if (tempBag != null){
          /*if the Bag is in the same place as the Hole,
          call the Hole's collision with the
          Bag as a parameter*/
          if ((tempBag.getGridX()==h.getGridX())
            &&(tempBag.getGridY()==h.getGridY())){
            h.collision(tempBag);
            bt.BagsArray[i]=null;
          }
        }
      }
      try {
        Thread.sleep(1);
      } catch (InterruptedException f){}
    }
  }//END run method
}//END class CollisionTracker

class BagsTracker extends Thread {
  GameEngine parent;
  Bag[] BagsArray;
  boolean alive;

  BagsTracker(GameEngine p){
    parent = p;
    BagsArray = new Bag [parent.getMaxBags()];
    alive = true;
  }//END constructor
  public void run (){
    while (alive){
      create();
      try {
          Thread.sleep(5000);
      } catch (InterruptedException e){}
      /*shouldn't hard code but sleepTime's for Bags*/
    }
  }//END run method

  void create(){
    for (int i=0; i<parent.getMaxBags(); i++){
      if (BagsArray[i]==null){
        BagsArray[i]=new Bag(parent);
        BagsArray[i].start();
        return;
      }
    }
  }//END create method

  Bag getBagAt (int i){
    return BagsArray[i];
  }//END getBagAt method

  void kill(){
    /*kill all Bags first*/
    for (int i=0; i<parent.getMaxBags(); i++){
      if (BagsArray[i]!=null){
        BagsArray[i].kill();
      }
    }
    alive = false;
  }//END kill method
}//END class BagsTracker

class Bag extends Thread {
  GameEngine parent;
  int direction, apples, bananas, price, gridX, gridY;
  boolean alive;
```

```
Bag (GameEngine p){
  parent = p;
  /*if the Trolly is in the left hand side of the
  maze, put the Bag on the right*/
  if (parent.Trolly.getGridX()
       <(int)parent.getGridWidth()/2)
    gridX = parent.getGridWidth()-1;
  /*if the Trolly is in the right hand side of the
  maze or in the middle, put the Bag on the left*/
  else gridX = 0;
  gridY = (int)parent.getGridHeight()/2;
  direction = (int)(Math.random() * 4);
  apples = (int)(Math.random()*6);
  bananas = (int)(Math.random()*6);
  price = apples*parent.getApplePrice()
           + bananas*parent.getBananaPrice();
  alive = true;
}//END Constructor

void move (){
  Square loc = parent.getSquareAt(gridX,gridY);
  if ((direction==0)&&(!loc.getNorth())) gridY++;
  else if ((direction==1)&&(!loc.getEast())) gridX++;
  else if ((direction==2)&&(!loc.getSouth())) gridY--;
  else if ((direction==3)&&(!loc.getWest())) gridX--;
  else {direction = (int)(Math.random()*4); move(); }
  parent.parent.repaint();
}//END move method

void kill (){
  alive = false;
}//END kill method

public void run (){
  while (alive){
    move();
    try {
      Thread.sleep(parent.getSleepTime()*10);
    } catch (InterruptedException e){}
  }
}//END run method
```

```
  int getGridX (){
    return gridX;
  }//END getGridX method
  int getGridY (){
    return gridY;
  }//END getGridY method
  int getPrice (){
    return price;
  }//END getPrice method
  int getApples (){
    return apples;
  }//END getApples method
  int getBananas (){
    return bananas;
  }//END getBananas method
}//END Class Bag

class Square {
  boolean north, south, east, west;
  Square (){
    north = false;
    south = false;
    east = false;
    west = false;
  }//END constructor

  void setNorth (boolean value){
    north = value;
  }//END setNorth method
  void setSouth (boolean value){
    south = value;
  }//END setSouth method
  void setEast (boolean value){
    east = value;
  }//END setEast method
  void setWest (boolean value){
    west = value;
  }//END setWest method

  boolean getNorth (){
    return north;
  }//END getNorth method
  boolean getSouth (){
    return south;
  }//END getSouth method
  boolean getEast (){
    return east;
  }//END getEast method
  boolean getWest (){
    return west;
  }//END getWest method
}//END Class Square
```

A.2 Game.java

```
import java.applet.Applet;
import java.awt.Graphics;
import java.awt.Color;
import java.awt.*;
import java.awt.event.*;
import java.util.*;
import javax.swing.*;


public class Game extends Applet {
  GameEngine engine;
  static GamePanel panel;
  int columnWidth;

  public void init (){
    /*it's important that columnWidth is set before the
    Panel constructor*/
    columnWidth = 20;
    engine = new GameEngine(this);
    panel = new GamePanel(this);
    add (panel);
  }//END init method

  public void start (){
  }//END start method
  public void stop (){
  }//END stop method
  public void destroy (){
    engine.BagsTracker.kill();
    engine.collisionTracker.kill();
  }//END destroy method

  public void paint (Graphics g){
    panel.maze.repaint();
  }//END paint method

  int getColumnWidth (){
    return columnWidth;
  }//END getColumnWidth method

  static void updateDisplay (String textIn){
    panel.display.append(textIn);
  }//END updateDisplay method

  void setEditInput (boolean b){
    panel.input.setEditable(b);
  }//END setEditInput

  void reset (){
    engine.reset();
  }//END reset method
}//END class Game

class GamePanel extends Panel implements ActionListener {
  Game parent;
  Maze maze;
  Button reset;
  Display display;
  Input input;

  GamePanel (Game p){
    parent = p;
    setLayout(new BoxLayout(this,BoxLayout.Y_AXIS));
    maze = new Maze (this);
    maze.setSize(11*parent.getColumnWidth()
                  +20,11*parent.getColumnWidth()+20);
    reset = new Button ("Reset the Game");
    display = new Display ("", 5, 20, 1);
    input = new Input (this, 20);
    reset.addActionListener(this);
    add(maze);
    add(reset);
    add(display);
    add(input);
  }//END constructor

  int getColumnWidth (){
    return parent.getColumnWidth();
  }//END getColumnWidth method

  public void actionPerformed (ActionEvent e){
    parent.reset();
  }//END actionPerformed
}//END class GamePanel

class Maze extends Canvas implements KeyListener {
  GamePanel parent;
  int columnWidth;
  GameEngine engine;

  Maze (GamePanel p){
    addKeyListener(this);
    parent = p;
    columnWidth = parent.getColumnWidth();
    engine = parent.parent.engine;
  }//END constructor

  public void paint (Graphics g){
    /*draw rectangle around the wHole maze*/
    g.drawRect(10,10,11*columnWidth,11*columnWidth);
    g.setColor(Color.black);
    /*draw in Checkout and Exit*/
    g.drawString("C",(columnWidth
                  +engine.Checkout.getGridX()
                  *columnWidth)-5, 10*columnWidth+25
                  -(engine.Checkout.getGridY()
                  *columnWidth));
    g.drawString("E", (columnWidth
                  +engine.Exit.getGridX()
                  *columnWidth)-5, 10*columnWidth+25
                  -(engine.Exit.getGridY()
                  *columnWidth));
    /*fill in square with Hole in it*/
    g.fillRect(columnWidth+engine.Hole.getGridX()
               *columnWidth-10, 10*columnWidth+10
               -(engine.Hole.getGridY()*columnWidth),
               columnWidth,columnWidth);
    /*draw a green rectangle around
    the apples number in the Trolly*/
    g.setColor(Color.green);
    g.fillRect(columnWidth-10+engine.Trolly.getGridX()
               *columnWidth, 10*columnWidth+30
               -(engine.Trolly.getGridY()*columnWidth)
                -columnWidth, columnWidth/2, columnWidth);
    g.setColor(Color.yellow);
    /*draw a yellow rectangle around
    the bananas number in the Trolly*/
    g.fillRect(columnWidth+engine.Trolly.getGridX()
               *columnWidth, 10*columnWidth+30
               -(engine.Trolly.getGridY()*columnWidth)
               -columnWidth, columnWidth/2, columnWidth);
    /*draw a red rectangle around the Trolly*/
    g.setColor(Color.red);
    g.drawRect(columnWidth-10+engine.Trolly.getGridX()
               *columnWidth, 10*columnWidth+30
               -(engine.Trolly.getGridY()*columnWidth)
               -columnWidth, columnWidth, columnWidth);
    /*draw in numbers on Trolly*/
    g.setColor(Color.black);
    g.drawString(""+engine.Trolly.getApples(),
                  columnWidth-10+engine.Trolly.getGridX()
                  *columnWidth, 10*columnWidth+25
                  -(engine.Trolly.getGridY()*columnWidth));
    g.drawString(""+engine.Trolly.getBananas(),
                  columnWidth+engine.Trolly.getGridX()
                  *columnWidth, 10*columnWidth+25
                  -(engine.Trolly.getGridY()*columnWidth));
    for (int i=0; i<engine.getMaxBags(); i++){
```

```
    /*if the Bags not null draw the same as for
    the Trolly but blue outline*/
    if (engine.BagsTracker.getBagAt(i)!=null){
      g.setColor(Color.green);
      g.fillRect(columnWidth-10+
         engine.BagsTracker.getBagAt(i).getGridX()
         *columnWidth, 10*columnWidth+30
         -(engine.BagsTracker.getBagAt(i).getGridY()
         *columnWidth)-columnWidth, columnWidth/2,
         columnWidth);
      g.setColor(Color.yellow);
      g.fillRect(columnWidth+
         engine.BagsTracker.getBagAt(i).getGridX()
         *columnWidth, 10*columnWidth+30
         -(engine.BagsTracker.getBagAt(i).getGridY()
         *columnWidth)-columnWidth, columnWidth/2,
         columnWidth);
      g.setColor(Color.blue);
      g.drawRect(columnWidth-10
         +engine.BagsTracker.getBagAt(i).getGridX()
         *columnWidth, 10*columnWidth+30
         -(engine.BagsTracker.getBagAt(i).getGridY()
         *columnWidth)-columnWidth, columnWidth,
         columnWidth);
      g.setColor(Color.black);
      g.drawString(""
         +engine.BagsTracker.getBagAt(i).getApples(),
         columnWidth-10
         +engine.BagsTracker.getBagAt(i).getGridX()
         *columnWidth, 10*columnWidth+25
         -(engine.BagsTracker.getBagAt(i).getGridY()
         *columnWidth));
      g.drawString(""
         +engine.BagsTracker.getBagAt(i).getBananas(),
         columnWidth
         +engine.BagsTracker.getBagAt(i).getGridX()
         *columnWidth, 10*columnWidth+25
         -(engine.BagsTracker.getBagAt(i).getGridY()
         *columnWidth));
    }
  }
```

```
    g.setColor(Color.black);
    /*up as far as ten, don't need outside line*/
    /*draw a line to the east for
    every Square that should have one*/
    for (int i=0; i<10; i++){
      for (int j=0; j<11; j++){
        if (engine.getSquareAt(i,j).getEast()){
          g.drawLine(10+columnWidth+i*columnWidth,
            11*columnWidth+10-j*columnWidth,
            10+columnWidth+i*columnWidth,
            10*columnWidth+10-j*columnWidth);
        }
      }
    }
    /*starting from one, don't need outside line*/
    /*draw a line to the south for
    every Square that should have one*/
    for (int i=0; i<11; i++){
      for (int j=1; j<11; j++){
        if (engine.getSquareAt(i,j).getSouth()){
          g.drawLine(10+i*columnWidth,
            11*columnWidth+10-j*columnWidth,
            columnWidth+10+i*columnWidth,
            11*columnWidth+10-j*columnWidth);
        }
      }
    }
  }//END paint method

  public void keyPressed (KeyEvent e){
    int code = e.getKeyCode();
    if (code == 37){
      engine.Trolly.moveWest();
    }
    if (code == 38){
      engine.Trolly.moveNorth();
    }
    if (code == 39){
      engine.Trolly.moveEast();
    }
    if (code == 40){
      engine.Trolly.moveSouth();
    }
  }//END class keyPressed
  public void keyTyped (KeyEvent e){
  }//END class keyTyped
  public void keyReleased (KeyEvent e){
  }//END class keyReleased
}//END class Maze

class Display extends TextArea {
  Display (String text, int rows, int columns, int scroll){
    super (text, rows, columns, scroll);
    setEditable (false);
  }//END constructor
}//END class Display

class Input extends TextField implements ActionListener {
  GamePanel parent;
  GameEngine engine;

  Input (GamePanel p, int columns){
    super (columns);
    parent = p;
    engine = parent.parent.engine;
    setEditable (false);
    addActionListener(this);
  }//END constructor

  public void actionPerformed (ActionEvent e){
    int in = 0;
    try {
      in = Integer.parseInt(getText());
      parent.display.append("You guessed "+in+"\n");
      /*it shouldn't be possible to get here if
      the condition on the following if statement
      is false but it's in for safety's sake*/
      if (engine.gameEnd != null){
        engine.gameEnd.check(in);
      }
    } catch (NumberFormatException f){}
    setText("");
  }//END actionPerformed
}//END class Input
```

A.3 Game.html

```
<HTML>
<HEAD>
<TITLE> The Shopping Game </TITLE>
</HEAD>
<BODY>
<a href="rules.html">Read the rules</a>
</br>
</br>
<APPLET CODE="Game.class" WIDTH=250 HEIGHT=400>
</APPLET>
</BODY>
</HTML>
```

A.4 Rules.html

```
<HTML>
<HEAD>
<TITLE> Rules for the Shopping Game </TITLE>
</HEAD>
<BODY>

The object of the Shopping Game is to work out how much
one apple costs and how much one banana costs and get
out the Exit.</br>
</br>
You play as a Trolly which has a red box around it,
starts off in the bottom left corner of the maze and
starts with no apples (the number on the green side)
and no bananas (the number on the yellow side).
If you click on the maze you will be able to move the
Trolly around with the arrow keys.</br>
</br>
The maze also has Bags moving around it.  These Bags
```

```
have blue box around them and a number of apples (on
the green side) and bananas (on the yellow side). When
the Trolly runs over a Bag it collects it and adds it
to what's already in the Trolly.  Remember some Bags
are more useful than others!  Be careful though, if the
Bags reach the Hole in the centre of the maze they'll
fall down it and they won't come back.</br>
</br>
You can go and see how much the things you have already
collected are worth by going to the Checkout, which is
in the bottom right hand corner of the maze.  This will
tell you how many apples and bananas you have and how
much they are worth together.</br>
</br>
When you think you know the answer you can go to the
Exit and try to get out.  The Exit will ask you how
much one apple costs and how much one banana costs.
You need to click in the input box below and input the
numbers from your keyboard to guess.  If you guess
right, the game will restart with new prices.  If you
guess wrong, you will need to move away from the Exit
and back before you can make another guess.</br>
</br>
</br>
Have fun!
</br>
</br>
<a href="Game.html">Back to the Game</a>

</BODY>
</HTML>
```

Appendix B

Log

B.1 Design and Implementation of Final Game

Week: Jan 31st - Feb 6th:

Implemented original design. Realised that de sign would need to be made based on testing. Maze needed to be redesigned, speed of the Bags needed to be fixed and game needed to repaint when Bags moved. Realised that the 'kill' method of BagsTracker would need to kill all the Bags and implemented this. Added in hack of 'Repainter' class in back end that extends class and sits there repainting the applet the whole time. Probably won' t leave this in long term but it makes it work for now.

Week: Feb 7th - Feb 13th:

Realised that apple and banana prices would always be set to zero because of error in code and fixed it. Added limits for the number of apples and bananas and methods for returning them to GameEngine. Considered moving these limits to Trolly. Started working on collisions. Added collision tracker class and instanciation in GameEngine. Added collision methods to Trolly, Checkout, Hole and Exit. Added some functionality to Trolly's collision method, got other collision methods to print to the screen to insure they were being called before coding functionality. Added a method to BagsTracker class that would return the Bag at a given place in the array, this was to stop direct referencing into array. Updated Bag's move method so that repaint would always be called. Added call to repaint in all moves and all collisions. Added parent Game to GameEngine class. Realised that player could sit on Bag spawn point so added that Bags appear on opposite side of maze from Trolly. After making design decision to empty the Trolly when the player went to the Checkout, added reset method to Trolly which set apples, bananas and price to zero to facilitate this and called it from the collision between the Checkout and the Trolly. Also added method to Trolly that would return true if the Trolly was empty so that we could only call the collision between the Trolly and the Checkout if the Trolly was empty, to solve the problem of the collision being called repeatedly which I had been aware of for a while. Solved similar problem at Exit with 'atExit' boolean in Trolly, the collision will only be called if the Trolly isn't already at the Exit. This involved a little more work because I had to add code in to set it to false when it moved away as well. Also added code that will set the input textfield to editable while the Trolly is at the Exit. Moved all debugging notes to the display within the game rather than the command line, doesn't really have an effect on the final game but useful for now. Added random generation for apple and banana prices (had been setting to 10 for testing). Added reset method to GameEngine. Replaced all occurences of Game.updateDisplay() with parent. parent.updateDisplay(). This only worked because the method was static but it doesn't fit with the design to call it that way. Added GameEnd class, instance thereof in GameEngine and calls to create and start it in the collision of Exit. It extends Thread and has a higher priority than the other threads, so should hopefully stop them from working, once the rest of its functionality is in. It' s going to basically sit and wait until it gets input from the user. Changed initial direction of Bags to random. The Bags were set to start with a direction of either South or West because they originally started in the North-East courner of the maze. Could have changed this to North or South because those will be the only directions available but this may change again. Removed all calls to change whether input is editable or not from everywhere except GameEnd class because it' s where they should be called from and it makes it much easier to keep track. Updated run method in 'GameEnd' class so that it sets input to editable, waits for inputs (or the Trolly to move away) and sets it to not editable. Added kill collisionTracker bit to destroy of Game applet. This is important because collisionTracker won't end by itself. GameEngine's not an applet, so it doesn't need to be killed. Increased size of everything. Removed repainter class and replaced it by having GameEngine in backEnd take 'this' as a parent and referring to it. Removed calls to repaint from key press method, and added them to Trolly instead. Did this because it's really the fact that the Trolly has moved that's causing the repaint, not the fact that the key was pressed. Added variable columnWidth to reduce hardcoding and make the size of the maze more variable. Replaced displaying B and T for Bags and Trollys with displaying the number of apples in green and the number of bananas in yellow. Changed background colour to make this slightly more visible. Started to implement new front end design. Added all the pieces and got paint to work. Added keylistener to maze. Made panel static because it needs to be referenced from a static context. Added updateDisplay method which takes a String and appends it to the display, this will be called by object in the GameEngine. Did this to stop things in the GameEngine having to refer to things in things in the Game class instead of just to the Game class. Basically this helps to keep the front and back ends as separate as possible. Changed width of Display and Input text components. Added setEditable method to GamePanel which just sets the editable status of input, again to keep front end and back end separate. Added code to actionPerformed method in Input which checks if input is a number and clears the text field either way. At the moment it just outputs 'you guessed' and the number to the display. Reduced the size of the maze so all the componants fit on the screen at once in the webpage. Added code to input that checks against gameEnd to see if we're looking for apples or oranges and compares the input to the relevant price and outputs the relevant thing to the display.

Week: Feb 14th - Feb 20nd:

updated run of GameEnd so that it always sets the editable value of Input to false because of issue found with testing that otherwise if you won it could stay editable. Reduced maximum number of apples and bananas the Trolly can hold to 9 of each to make the Trolly easier to draw. Updated message when player guesses wrong for more detail. Added code to display the time from the start of the game to the current time when the player wins. Also removed line that set priority as there was no need for it and it would have been a hacky way of doing what I was trying to do even if it had worked, which it hadn't. Have concluded that what I was trying to do (stop the other threads while the Trolly was in the Exit) was not necessary anyway. Removed all references to parent.parent.engine and replaced them with calls to local 'version' of engine. This involved added the local version to Input. This was done again to keep the front end and back end separate. Replaced all calls in Game.java to BagsTracker.BagsArray[i] with BagsTracker.getBagAt(i) as this is much neater and helps to keep back end and front end separate. Changed dimensions of where stuff in Trolly is drawn so it has the same shape as the Bags. Replaced drawing H in the centre with drawing a black rectangle. Changed color of Exit and Checkout to black. Changed paint method of maze so that it's white, with black numbers on green and yellow rectangles with a red square for the Trolly and a blue square for the Bags. Added comments to show what's going on. This is not as scalable as the last version. Fixing this scalability is not a priority. Changed layout of panel to BoxLayout. Had to include javax.swing.* to do this. Added reset button to GamePanel class and had GamePanel listen for events for it. Added reset method to Game class which calls the reset in engine and made call to it from actionperformed in Panel which is listening for presses of the reset button. Moved everything up and to the left a little for prettiness sake.

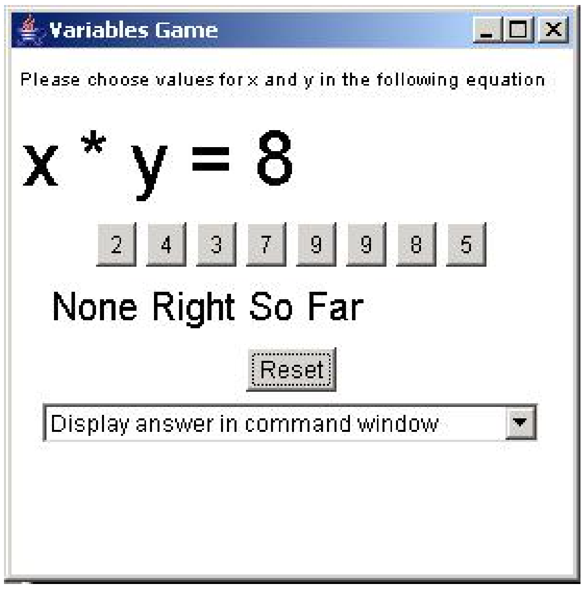
Week: Feb 28th - Mar 6th:

Moved checking of whether or not they were right from front end to back end because it makes more sense.

Appendix C

Variables Game

Put in bit here about how what comes after is the screenshot and the code for the game I worked on before Christmas. Talk about where it's referenced in the rest of the write up.



Please choose values for x and y in the following equation

x * y = 8

None Right So Far

Display answer in command window

```
import java.awt.*;
import java.awt.event.*;

class VariablesFrameChoice extends Frame
  implements ActionListener, WindowListener, ItemListener {
  int numOptions;
  int upperBound;
```

```
Variables question;
Label eqn;
Label blurb;
Label numRight;
OptionButton options[];
String eqnText;
Button reset;
Choice display;
boolean displayTrue;
/*this bit is hardcoded for two variables*/
boolean xRight;
boolean yRight;

VariablesFrameChoice
  (String s, int numOptionsIn, int upperBoundIn) {
  super(s);
  setSize(300,300);
  setLayout(new FlowLayout());
  numOptions = numOptionsIn;
  upperBound = upperBoundIn;
  question = new Variables(numOptions, upperBound);
  blurb = new Label
    ("Please choose values for x and y"
    + " in the following equation");
    blurb.setFont(new Font
    ("TimesRoman 10 point.",40,10));
  eqn = new Label("     x " + question.operator
    + " y = " + question.z + "      ");
    eqn.setFont(new Font
      ("TimesRoman 14 point bold.",300,40));
  numRight = new Label("    None Right So Far     ");
    numRight.setFont
      (new Font ("TimesRoman 12 point bold.",80,20));
  options = new OptionButton [numOptions];
  reset = new Button("Reset");
  xRight = false;
  yRight = false;
  displayTrue = false;
  display = new Choice();
  add(blurb);
  add(eqn);
```

```
    for (int i=0; i<numOptions; i++) {
      options[i] = new OptionButton(question.options[i]);
      add(options[i]);
      options[i].addActionListener(this);
    }
    add(numRight);
    add(reset);
    reset.addActionListener(this);
    addWindowListener(this);
    display.add("Display answer in command window");
    display.add("Don't display answer in command window");
    add(display);
    display.addItemListener(this);
    setVisible(true);
  }//END VariablesFrame constructor


  public void actionPerformed(ActionEvent event) {
    if ((event.getActionCommand()).equals("Reset")) {
      question = new Variables(numOptions, upperBound);
      eqn.setText(" x " + question.operator
        + " y = " + question.z + "  ");
      numRight.setText("None Right So Far");
      xRight = false;
      yRight = false;
      if (displayTrue) {
        System.out.println("" + question.x
          + question.operator + question.y + "="
          + question.z);
      }
      for (int i=0; i<numOptions; i++) {
        (options[i]).setLabel(""+question.options[i]);
      }
    }
    /*
    at the moment lets them click something and puts it
    in where it should be if it's right for either.
    I will redo this.
    */
    else {
      if (Integer.parseInt
       (event.getActionCommand()) == question.x) {
        xRight = true;
        if (yRight) {
          eqn.setText(" " + question.x + " "
            + question.operator
            + " " + question.y + " = " + question.z);
          numRight.setText("All Right!");
        }
        else {
          eqn.setText(" " + question.x + " "
            + question.operator
            + " y = " + question.z);
          numRight.setText("One Right So Far");
        }
      }

      else if (Integer.parseInt
       (event.getActionCommand()) == question.y) {
        yRight = true;
        if (xRight) {
          eqn.setText(" " + question.x + " "
            + question.operator
            + " " + question.y + " = " + question.z);
          numRight.setText("All Right!");
        }
        else {
          eqn.setText
          (" x " + question.operator + " " + question.y
           + " = " + question.z);
          numRight.setText("One Right So Far");
        }
      }


    }
  }

  public void windowClosed(WindowEvent event) {
    System.out.println("window closed");
  }//END method windowClosed
  public void windowDeiconified(WindowEvent event) {
    System.out.println("window deiconified");
  }//END method windowDeiconified
  public void windowIconified(WindowEvent event) {
    System.out.println("window iconified");
  }//END method windowIconified
  public void windowActivated(WindowEvent event) {
    System.out.println("window activated");
  }//END method windowActivated
  public void windowDeactivated(WindowEvent event) {
    System.out.println("window deactivated");
  }//END method windowDeactivated
  public void windowOpened(WindowEvent event) {
    System.out.println("window opened");
  }//END method windowOpened

  public void windowClosing(WindowEvent event) {
    System.out.println("window closing");
    System.Exit(0);
  }//END method windowClosing

  public void itemStateChanged (ItemEvent e) {
    displayTrue = !displayTrue;
    //String s = e.getItem().toString();
    //System.out.println("You prefer : " + s);
  }//END itemStateChanged method

}//END CLASS VariablesFrame

class OptionButton extends Button {
  int value;

  OptionButton (int valueIn) {
    super ("" + valueIn);
    value = valueIn;
  }//END OptionButton constructor

}//END CLASS OptionButton
```

```
public class VariablesGameChoice {
  public static void main(String args[]) {
     //create a VariablesFrame calling constructor
     VariablesFrameChoice variablesFrameInstance
     = new VariablesFrameChoice
     ("Variables Game",8,10);
  } //END main method
} //END CLASS VariablesGame
```

References

[1] Jennifer Vesperman, *Essential CVS*, O'Reilly Publications

[2] The Source Forge .net website *http://sourceforge.net/*

[3] Mikado, the Maynooth University Online Community *http://www.mikadosoc.ie/*

[4] NUIM Maynooth Games Society *http://www.minds.may.ie/∼games/*

[5] The Irish Gaming Website *http://www.irishgaming.com/*

[6] Bruce Eckel, *Thinking in Java*, Prentice Hall books; ISBN: 0-13-659723-8

[7] Kathy Walrath, Mary Campione, Alison Huml, Sharon Zakhour, *The JFC Swing Tutorial*, Sun Microsystems

[8] John Walkenbach, *Excel 2000 Bible*, IDG Books Worldwide; ISBN: 0-7645-3259-6

[9] Terry Quatrani, *Visual Modeling With Rational Rose and UML*, Booch, Jacobson, Rumbaugh; ISBN: 0-2017-2932-6