G52GRP Final Report

EduCraft
Minecraft as a Collaborative Learning Tool

Group `gp13-pxb1`:
Tosin Afolabi (`ooa02u`)
Janos Bana (`jxb12u`)
Eze Benson (`exb02u`)
Ali Kerr (`axk02u`)
Ian Knight (`iak12u`)
Ying Wang (`yxw03u`)

April 4, 2014

Contents

Chapter 1

Introduction and Background

The EduCraft project is concerned with developing a series of extensions[1] for the popular computer game Minecraft, aimed at promoting collaborative learning in primary schools, particularly with reference to numeracy education. In this first section, some of the advantages of collaborative learning are explored in contrast to conventional teaching methods, and the game of Minecraft is introduced in the context of education.

1.1 Collaborative learning

> "Group interaction allows students to negotiate meanings, to express themselves in the language of the subject and to establish a more intimate and dialectical contact with academic and teaching staff than formal methods permit." [4, p. 1]

Group work has long been recognised as a key component in any form of successful education—it enables students to become more involved in their own work, and to engage with it in a different manner from what is normally seen in classrooms. Johnson and Johnson observe that "The human species seems to have a *cooperation imperative*", and that cooperation plays a central role in most areas of life [5, p. 12].

Collaborative or cooperative learning is contrasted with two other 'goal structures': competitive learning, and individualistic learning. Competitive learning consists of activities which set the students tasks which some are expected to 'win', by getting the highest score or completing the activity in the shortest time, for example. Individualistic learning is learning where there is no interaction between students at all—no comparisons are made, and students are solely concerned with working by themselves.

Johnson and Johnson argue that each of these goal structures has a role to play in education, but that each is suitable for different purposes [5]. Collaborative learning, specifically, is to be used in situations where the material being taught is especially complex or conceptual. This

[1]These are commonly called 'mods'. Throughout the report, we will refer to our product as 'the mod'.

makes it ideal for use in maths lessons: Yackel, *et al.* describe the benefits of group work in teaching maths to second-grade[2] students in the US [23].

1.2 Minecraft

> "Minecraft is a game about breaking and placing blocks. At first, people built structures to protect against nocturnal monsters, but as the game grew players worked together to create wonderful, imaginative things." [19]

Minecraft was initially developed in 2009, as a game where players could explore a randomly-generated world and place blocks to build structures. From the outset, Minecraft was not designed as a game that could be 'won'—in game industry terms, it was intended to be a 'sandbox' game where players set their own goals.

There have been a number of articles written exploring the potential use of Minecraft in supporting education [1], [20]. Habgood has already established the usefulness of computer games in general in education [3]; the question is whether Minecraft can be used in the ways that he suggests. This is what EduCraft seeks to establish, at least tentatively.

The official Minecraft Wiki has its own page dedicated to describing possible ways of using Minecraft in education [11], and suggests using the game's in-built system of building items:

> "The crafting system can help in teaching basic math (e.g. I need 3 Sugar Cane for Paper), which transitions to multiplication (I need 3 Paper and 1 Leather for a Book, and 3 Books for a Bookshelf, so I need 9 Paper and 3 Leather all together) and division (When I create Paper I get 3 at once, so $9/3 = 3$ times per Bookshelf I'll have to create Paper)."

We intend to build something more overtly mathematical than this basic concept, and the remainder of the report describes the requirements that have been set for the project, along with a record of our initial design and prototypes.

[2] Roughly equivalent to year 5 (ages 7–8) in the UK

Chapter 2

Requirements Specification

2.1 Functional requirements

1. The game must cater for users who are currently at level 2 of Key Stage 2 mathematics.
    - 1.1 There must be a puzzle in the game which challenges the users knowledge of subtraction facts up to 10.
    - 1.2 There must be a puzzle in the game which challenges the users knowledge of number ordering from 0 - 100.
    - 1.3 There must be a puzzle in the game which challenges the users knowledge of number sequencing. This must include odd and even numbers.
2. The game must cater for users who are currently at level 3 of Key Stage 2 mathematics.
    - 2.1 There must be a puzzle in the game which challenges the users knowledge of the 2,3,4,5 and 10 times table.
    - 2.2 There must be a puzzle which forces a user to use multiplication or division to solve it. The multiplication or division must only be used with whole numbers. When choosing numbers for division numbers with remainders are to be included.
    - 2.3 The game should include puzzles which involves breaking entities into simple fractions.
    - 2.4 Within the game there should be puzzles which require users to equate simple fractions.
3. The game must cater for users who are currently at level 4 of Key Stage 2 mathematics.
    - 3.1 Puzzles must exist within the game which can only be solved using division by 10 or 100. The entities being divided must be whole numbers.
    - 3.2 A puzzle must exist which tests users upon their knowledge of multiplication facts up to 10x10. This puzzle should then go on to test whether the user can quickly derive correct division facts.
    - 3.3 Within the game there should be a puzzle which involves adding and subtracting entities which represent decimal numbers to 2 decimal places.
    - 3.4 A puzzle should exist which forces the user to order decimal numbers. These decimal numbers can be up to three decimal places in length.

4. Levels in the game must be constructed in such a way that each user contributes to a problem before all the users can progress. A user must contribute within the game physically.
5. The game must not be single player and should allow players in groups of 3-4 to play together.
6. The players should be able connect to dedicated servers to play the game.
7. The mod must implement coins as one of the visual representations of numbers.
8. The mod must have custom weapons which can be used to defeat zombies.
9. Coins and other visual representations of numbers must be collected by defeating zombies.
10. Zombies must not drop numbers or coins if the user does not defeat them using the specific mathematical weapon.
11. Zombies must have a number above their head clearly visible to the player indicating what number they drop upon being defeated.
12. Zombies should drop a number or coins equal to the number stated above the zombie's head.
13. Players must be able to craft all mathematical operators using items collected throughout the level. (Mathematical operators being +,-,*,/)
14. Each mathematical operator must be represented visually in a way which is commonly understood by people of the specified age group.
15. Puzzles must be solved in chambers as well as open world spaces.
16. Players should not be allowed to progress until the puzzle they are currently on is completed.
17. Mathematical operators must be usable objects and not weapons.
18. Mathematical operators will be crafted using items which have been ascertained through resource collecting, where resource collecting is the action of getting items in any way but defeating zombies.
19. Equations must be able to be crafted using mathematical operators and numbers collected from defeated zombies.

2.2 Non-functional requirements

1. The game must be implemented in Minecraft through a mod.
2. The mod should work on Minecraft version 1.6.4.
3. The mod must work on the windows operating system.
4. The mod must work on a machine which has the minimum requirements of Minecraft or better.

5. The game should abide by the ethics code in place within the education system for those between the ages of 8 and 11.

Chapter 3

Design

3.1 Educational Design Decisions

3.1.1 Prototype 1 (Dummy Mod)

We decided to design most of the game around the concept of defeating enemies that are already in Minecraft. These enemies will drop numbers which the player can later use for progression. This feature was the main feature implemented at this stage of the prototype. We then began to design how this enemy would fit into the world, this is where we introduced the idea of the room. Using these two concepts in combination allowed us to force the player to perform a certain task because the player could not leave the room until the specified task was complete. Therefore we could force players to practice maths and later on we would use this concept to force collaboration also. However at this stage of the development we would only use these features to force the player to count coins just as proof that this concept was a viable one.

3.1.2 Prototype 2 (Second Stage)

In this prototype we wanted to continue using the features from the previous prototype and also introduce some others to make the game more about learning and practising maths. We based our prototype design upon the mathematics curriculum taught to the children of the mods target age group. This meant we would design a prototype which had simple mathematics such as addition,subtraction,division and multiplication. To implement these arithmetic operations we would require items the game of Minecraft does not naturally have, operators and numbers. The level would be designed in such a way that the player would have to create operators and get numbers by killing zombies. These two items would somehow be used together in an arithmetic operation and produce a result, the player would then use this result to complete the level. The next problem faced was that if we have the two custom items we would also need to implement a custom way to combine the two items and calculate a result depending on what those two items were. We decided at the time that researching and finding a natural construct in Minecraft was the best solution for two reasons. Firstly it would be better to reuse code that is already written and proven to work well and secondly the player is playing Minecraft therefore we should use as many features already in Minecraft to keep the player feeling as if they are actually playing Minecraft.

The final prototype design feature we decided to add was another constraint. We did not want the player to have an infinite amount of attempts at the problem, if we let the player do this the player would eventually choose the right operator or combination of operators and numbers required to solve the puzzle. This defeats the entire purpose of the game since the player would not practice maths or learn anything new they would have just guessed their way to the answer. Therefore we decided that when the prototype was implemented we would either limit the amount of operators they could create or the amount of numbers they could collect. Although this limits the amount of attempts the player can make at an arithmetic operation the player can still guess and get the right answer. Unfortunately this is something we cannot eliminate through level design decisions.

3.1.3 First Release

In this prototype we would now take the previous two prototype's features and add the final feature of collaborative learning. Up until this point the prototypes were designed so that one player could play, in this level we designed it for four players. At the start their would be the same features as previously such as only being able to progress after creating a certain number or operator. This was to get the players who had not played the mod or much Minecraft used to the game. We decided to implement this tutorial part of the level due to some testing we did with a participant. We realised that people need to be taught all of the constructs before they can really get any learning or practice done. We would then design the next stage of the level with the most important feature of all collaborative learning. We decided that the team would be split up into two groups of two, the two groups would then have different resources but neither group would have enough resources to progress. The groups would then have to give each other the correct resources so that both groups could progress this would force each team to communicate and collaborate.

We then went on to add features which would comply with the curriculum we did this so that the children would have a wider variety of mathematics they could learn and it would also help hold the attention of the children. These features are mentioned more in depth in the level design subsection. The final design was based on the castle,pyramid and rook level designs.

3.2 Game Design

The aim of our game design was to efficiently encode collaborative learning in the mechanics of the game. This section will list and describe all the additional elements that we decided to add to the existing Minecraft 1.6.4 core game in order to create a playable mod that would fulfil our requirements. We gradually introduced new elements as and when required.

The mod is set to be played in adventure mode[1]. We have tried to preserve as much of the

[1]"Adventure mode is a game mode intended for player-created maps by limiting some of the gameplay in Minecraft, in which the player cannot directly destroy most blocks to avoid spoiling adventure maps or grieving

Minecraft concepts in our extension as possible, so that players familiar with the game can easily understand the mechanics of our mod. Inexperienced users will also get to know the Minecraft philosophy whilst using our extension and will be able to play the original game or some other mods without any problems in the future.

3.2.1 Items[2]

- **Numbers**
  These are the most important element in our extension and are crucial for promoting the practice of mathematics. Numbers are used in several of the features our mod offers.
- **Operators**
  Operators are as important as numbers, but only used in certain parts of the mod and are key elements of performing arithmetic operations.
- **Keys**
  Keys are used for progressing in the game. They allow us to place constraints on the free movement of players.
- **Maths Wands**
  Maths Wands enable us to limit the choice of weapons used for killing mobs.

3.2.2 Blocks[3]

- **Crafting Table**[4]
  - **Operator Bench**
    This is a slightly modified version of the Crafting Table and it is used for creating mathematical operators from sticks.
  - **Ordering Bench**
    This is a modified version of the Crafting Table which only accepts numbers as inputs. It requires the players to order the input numbers in order to generate an output. It has three different subtypes accepting all, only odd or only even numbers.
  - **Calculator Bench**
    This is a modified version of the Crafting Table which only accepts numbers and operators as inputs. It allows the players to generate new numbers from the inputs.

servers. Most blocks cannot be destroyed without the proper tools. However, players can still interact with mobs and craft items."[12]

[2] "Items are objects which only exist within the player's inventory and hands - which means, they cannot be placed in the game world. Some items simply place blocks or entities into the game world when used. They are thus an item when in the inventory and a block when placed."[16]

[3] "Blocks are the basic units of structure in Minecraft, and are essential to the gameplay. Together, they build up the in-game environment and can be utilized in various fashions."[13]

[4] "The Crafting Table (or Workbench) is one of the most essential blocks in Minecraft. Right-clicking on a crafting table placed in the world will open the 33 crafting grid which allows you to craft more items than with the crafting grid in your inventory. A crafting table is essential in order to progress at all in the game."[14]

It performs simple arithmetic operations based on the type of the operator in the input matrix.

3.2.3 Mobs[5]

– **Zombies**
  This modified version of the original Zombie hostile mob can only be killed by using Math Wands and they drop numbers when slayed.

– **Skeletons**
  This is a modified version of the original Skeleton hostile with similar properties to Zombies. Skeletons are armed with bows that shoot arrows at the player. This makes them more difficult to kill.

3.3 Level Design

There are no levels in Minecraft, so we decided to create different locations representing the levels. The locations will also be referred to as levels in the rest of this report. The levels are used for controlling the learning experience of the players. Levels are a form of constraint that we have managed to utilise to create a learning curve within the game.

Level design decisions were deliberately made to ensure collaboration between players as well as encouraging communication. Researches show that communication and collaboration are linked and can improve the process of learning within groups. Most of the advanced levels consist of a mixture of easy and hard tasks. The purpose of this variety is to make the levels hard enough to engage the players and let them enjoy playing without making the game too hard to progress which would eventually result in the players losing interest.

Even though our mod does not have a storyline which would add a fantasy context to the gameplay, our location designs are based on recognisable historical architectures. Our initial world was completely flat, but we have created various landscapes in the vicinity of our locations to improve the visual experience in the mod. We wanted to show that Minecraft has the potential to implement a wide range of design ideas.

3.3.1 Locations

– **Concept of a room**
  A room is an abstract concept, that we used throughout the development process. It is used for controlling the free movement of the players in the world and to synchronize the collaborative progress of the team or subteams. Some of the initial tests with children

[5] "Mobs are living, moving game entities. Generally, mobs are affected by the environment in the same ways as the player: they are subject to physics, and they can be hurt by almost all the same things that harm the player: Catching on fire, falling, drowning or suffocating, and of course being attacked with weapons."[17]

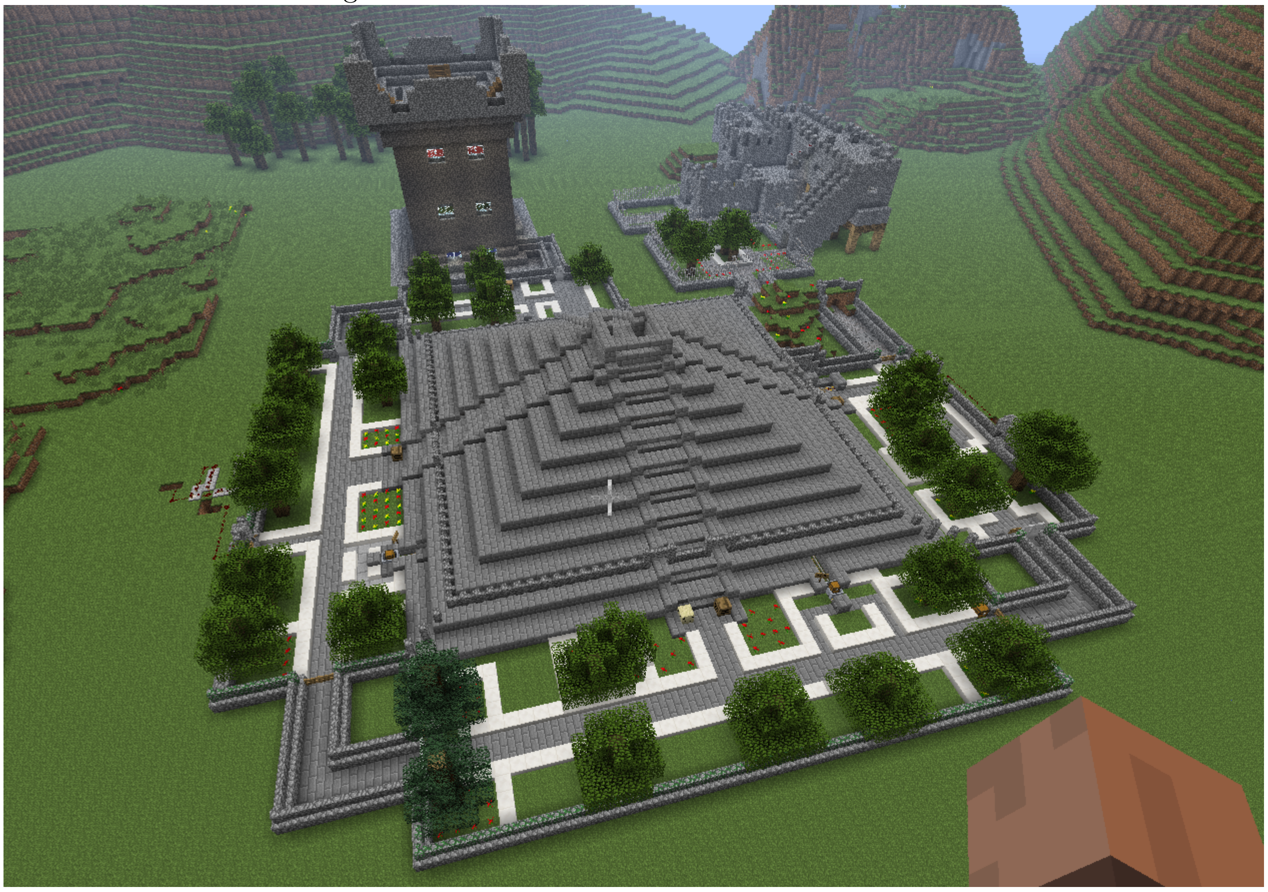
showed that they tend to walk around aimlessly to discover new parts of the world, which results in losing focus on the tasks to be completed. The room consists of an entrance and an exit. In most rooms once the player enters they cannot return to previous parts of the world. The room can only be exited once a specific task is completed.

- **Castle**
  The Castle requires players to collaborate and work as a group of four. This level is designed to teach the players how to play the mod and could also be considered as the tutorial level of the game. The players are introduced to the concept of a room and learn how to create operators, use various benches and obtain numbers.

- **Pyramid**
  The Pyramid level splits the players into two sub teams of two. There are various elements in this level that require good communication between the players in order to collaboratively progress through to the end of this level. There are limitations on the resources accessible to both teams, hence "no one can win without the help of the others". Ordering, subtraction and addition are the only mathematical operations required in this location.

- **Rook**
  The Rook implements the same principles as the Pyramid. The only difference is in the type of mathematical operations set as tasks, which are multiplication and division for this level.

Figure 3.1: Screenshot of the EduCraft world



Chapter 4

Implementation

4.1 Implementation of the mod

A full UML class diagram describing the mod is presented in Appendix A. In this chapter, we describe the decisions that were made when discussing how to implement those features.

- **Base Mod Class**[1]
  The Base Mod needs to contain the following line:

```
@SidedProxy(clientSide="tutorial.generic.client.ClientProxy",
    serverSide="tutorial.generic.CommonProxy")
public static CommonProxy proxy;
```

- **Proxy Classes**[2]

4.2 Implementation of Constraints

This section explains how we implemented the concept of the room and how we forced collaboration through level constraints. All levels used blocks, such as the Stone Brick block, which can not be broken with fists or an axe. This stops the player from breaking out of the level and skipping certain parts of the level without creating the correct item because we restricted the items the player could collect and create to ones which could not break through vital game blocks. We also put the game mode into adventure mode which means the player cannot take blocks from the creative tab or break any block instantly (both are possible when the player is in creative mode).

[1] "The base class is the class that Forge loads. All the classes in our mod have to be registered through this central location for the mod. It is advised to name this class after the name of the Mod. In our case it is EduCraft" [7]

[2] "Minecraft uses a client and server setup even on single player. The server side does all the maintaining of the world's state while the client renders the world. All code runs on both sides unless specified otherwise. The annotation `@SidedProxy` is used when we want the server to call the constructor of one class and the client the other. We use the proxy for registering images and hosting our GUI handler." [9]

4.2.1 Door Hopper Combination

To stop players from progressing without creating the correct numbers or keys we used a door and hopper combination. We put a hopper next to the door so that when the player thinks they have the correct item they can easily throw the item into the hopper and if it is correct the door will open. By the door (as well as underneath it) there are two hoppers on top of each other and a chest next to the top hopper. The top hopper is the hopper that the player can see [Show picture]. We then add the item that we want the player to throw, into each of the five slots in the bottom hopper. We then place five redstone powder trails down, four of which will be powered, if the player throws the right item in the top hopper it will drop the item into the bottom hopper and add it to the stack. If the wrong item is thrown into the top hopper it gets pushed into the chest. This is because we can only stack identical items in the same slot therefore if all slots are taken up in the bottom hopper we can only stack those item thus forcing the player to throw the correct item.

When the correct item is thrown into the hopper the fifth powder is then powered. We can then use a repeater to boost the weak signal and place a trail of powders leading to the block below the door. The door, when powered, will open. Doors become powered when the block they sit on is powered this is why we feed the redstone powder into the block below the door and not the door itself.

4.2.2 Piston Hopper Combination

We also devised another way in which we could stop the players from progressing only when they completed the specified task. We made stacked blocks so they were two blocks high; Minecraft players can only jump one block high when in adventure mode so they would not be able to jump onto these blocks. This then means if we put these blocks in the path of the player we could force the player to complete the task. Unfortunately this leaves a problem, we needed to find a new way to push or pull one of these blocks out of the way of the player. In this instance Ian suggested the use of the stick piston block. When the sticky piston block is powered by redstone magic it would push one block outwards (positioning is key and is shown well in the image) [insert image] meaning we could push a block next two block high stack or we could pull one of the blocks away. The redstone circuits required for either of these ideas are practically the same except one has a logical not in the redstone circuit.

In reality this idea is the same as the room method using the hopper and door combination but this idea also allows us to use blocks which would fit the theme of the level. As previously mentioned keeping immersion helps the player experience and makes them feel like theyre in the game as opposed to doing mathematics in school. Moreover the piston hopper combination allows us to create more unique levels and has a lot more practical applications than the door and hopper combination. Unfortunately the door is very limited because it has to be sitting on a block and must have open spaces infront and behind it and also it can only open in two direction. The piston hopper combination does not suffer from any of these limitations.

4.3 Implementation of Items

4.3.1 General Item Implementation Guide

Most item used within the mod are from the game itself but the most important items within the game have been custom made by us. To implement a custom item is fairly simple. Firstly we extend the Item class from the original Minecraft source files and then you implement the constructor which passes the itemID to the super constructor. From this point onwards anything else you implement in the constructor is optional and is totally dependant on the item you're creating. Most of the items we developed had fairly similar constructors and overridden methods.

4.3.2 Number Items

This, along with the operator item, is the most important item in the game. To implement it however is about as simple as any other item. Firstly we follow the general item implementation guide. We set the unlocalized name to number so that it is easy to use and find. Then we set its creative tab to the Educraft tab so we can find it easily within the game when building levels.

When then come to the methods we have the getUnlocalizedName method which is simple terms uses the already set unlocalizedname (number) and adds a dot followed by the number that object is representing. So for example number.35 would be the the unlocalizedName of the object representing 35. This function takes an ItemStack as its parameter this is because the actual number component of each object is encapsulated by metadata. Each item has metadata which holds the items damage value this can be changed for each number object so that this metadata holds is number value. We use a similar method to determine which icon each number object should have. The icon is chosen by taking the damage value and using that as the index of the icon you want in the icon array.

4.3.3 Operator Items

Once again all operators inherit from the item class. Therefore we implement the constructor in the same way as specified previously. Each construction sets the unlocalized name to a 3 letter representation of the operator plus the string operator. The maxStackSize is set to 4 so players can hold no more than 4 in a stack at a time. The creative tab the operators are put in is our custom educraft tab for convinience and finally we set the texture which is simply educraft: and the name of the operator. All operators implement the MathematicalOperator interface. The only method this interface forces is the getOperator method. This method simply returns what operator the current object is, we can use this later on in calculations because if we have two numbers and an operator object we dont know what operator it is we just know that its some kind of operator. In addition it means we then cannot calculate the result. The method will then counteract this problem and let us know what operator that object represents. The return type of the getOperator function is actually the type OperatorType which is an enumerated type.

So for example the multiplication type is represented by the name TIMES and contains two strings Multiplication and x. Every other operator type is represented in a similar way.

4.4 Implementation of Blocks

This section covers the most important steps that allowed us to create our own Crafting Tables with custom looks and specific behaviours.

During our initial attempts to create a modified Crafting Table we came across the problem of implementing our Numbers efficiently. We initially thought that we would have to hard code every single class for each Number item that could be used in the Calculator without having to create separate recipes. This stalled our progress for while, but luckily after a substantial amount of research we found a solution. This workaround allowed us to have one BaseNumber class and every time a new BaseNumber object is created a random metadata is assigned to that specific item. The metadata defines the value, the texture used and the on-screen text displayed for that Number item. After this issue was resolved we were able to proceed with the implementation of the Calculator.

Minecraft distinguishes between two different ways of handling information due to being a multi-player game, we will refer to these as Server Side and Client Side. The original Crafting Table only opens on the Client Side, which limits the interaction with this object to one player at a time. Initially we started with an implementation that only added custom looks and behaviours to our Crafting Tables. Later on, we recognised the importance of making the Crafting Tables Server Side to allow multiple players to interact with the same block simultaneously. This helped us to create an even more collaborative gaming experience.

4.4.1 Extended Block Types

The Ordering Table has three different types depending on the parity of the input numbers accepted by it. We have created an enum type named BenchType to describe the different types of Ordering Tables. OrderingTileEntity classes are created with an additional field differentiating the specific Ordering Table in the world when the block is created. This attribute will determine the behaviour of the Ordering Tables based on their type.

4.4.2 CraftingTileEntity

We created this class subsequent to our findings relating to the use of the Chest by multiple players simultaneously. We investigated how the Chest worked and found that Tile Entities are used to implement this feature.
Tile Entities are bound to specific coordinates in the world and their fields hold unique values. In the case of the Crafting Table, they hold the inventory of a Block. This allows interaction that can be seen by everyone over the network.
An additional field in our version of the Tile Entities keeps track of the number of players using

the Block.
The default constructor is not capable of setting up a crafting matrix (collection of input slots) as the CraftingTileEntity does not have a reference to a container on creation. Our work around was to initialise the CraftingTileEntity as soon as we create a container for a Block.

```
public synchronized CraftingTileEntity initialise(Container
   container) {
    if (this.container == null) {
        this.container = container;
this.craftMatrix = new InventoryCrafting(this.container, 1, 3);
        this.craftResult = new InventoryCraftResult();
    }
    incrUsers();
    return this;
}
```

4.4.3 GuiHandler

This class is essential for handling our custom-made GUIs. It implements the core IGuiHandler interface and has two methods:

– getServerGuiElement(...)
  generates a container, which forms the Server Side of The GUI

– getClientGuiElement(...)
  generates the GUI itself, which is displayed in the Client Side

4.4.4 Container[3]

The following are the changes we made to our extended version of the container to create a custom layout of the slot matrix:

– in the case of a Client Side only Crafting Table, field in class:

```
public InventoryCrafting craftMatrix = new InventoryCrafting(
    this, x, y);
```

– in the case of a Server Side Crafting Table, in constructor:

```
this.tileEntity = tileEntity.initialise(this, x, y);
```

[3] "The container is what connects the inventories of the player and tileentity to the GUI. The constructor defines the position on-screen and contents of each slot."[8]
x = the number of rows
y = the number of columns

– in any case, in constructor:

```
// adds output slot to the container
this.addSlotToContainer(new SlotCrafting(inventory.player,
   this.craftMatrix, this.craftResult, 0, 124, 35));

    // adds input slots to the container
    for (int l = 0; l < x; ++l) {
        for (int i1 = 0; i1 < y; ++i1) {
            this.addSlotToContainer(new Slot(this.craftMatrix
               , i1 + l * 3, 30 + i1 * 18, 17 + l * 18));
        }
    }
```

– in the case of a Server Side Crafting Table, in the onContainerClosed() method:

```
/**
 * Called whenever the container is closed. If no one is
    using the container
 * any more, then it should drop everything inside it, like a
     crafting
 * table.
 *
 * @param player
 *            the player who closed the container
 */
@Override
public void onContainerClosed(EntityPlayer player) {
    super.onContainerClosed(player);
    this.tileEntity.decrUsers();

    if (!this.worldObj.isRemote && !this.tileEntity.
       isBeingUsed()) {
            for (int i = 0; i < 3; ++i) {
                ItemStack itemstack = this.craftMatrix.
                   getStackInSlotOnClosing(i);

                if (itemstack != null) {
                    player.dropPlayerItem(itemstack;
                }
            }
```

x = the number of rows
y = the number of columns
Please note that in Slot(..., x, y, z) and SlotCrafting(..., x, y, z) the last three arguments passed in are slotIndex, xDisplayPosition and yDisplayPosition in this order. Coordinates are represented in pixels. The default size of a slot is 18x18 pixels.

```
        }
}
```

4.4.5 Gui

This class is responsible for the creation and display of GUIs. There are two methods within this class. One is responsible for drawing the background and the other one is for drawing the foreground.

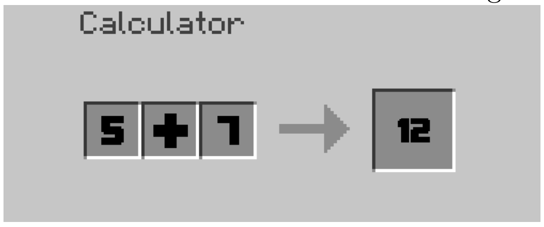
```
/**
* Standard prefix for every GUI texture created for the mod.
*/
public static final String GuiTexturePrefix = "educraft" + ":" +
   "textures/gui/";

private ResourceLocation calculator = new ResourceLocation(
   EduCraft.GuiTexturePrefix+ "FileName.png");
```

4.4.6 CraftingManager

In the Crafting Manager classes for the Calculator and Ordering Bench we simplified the operation of the core version of this class. This was possible as we are not using any Recipes for implementing the logic of how the inputs are checked and the way the output is generated. Both Crafting Tables have only 3 input slots and this allowed us to validate inputs in a simpler manner.

The logic implemented for the Calculator is to have two operands (instance of BaseNumber) in slots 1 and 3 and an operator (instance of MathematicalOperator) in slot 2 of the input matrix[9]. If this condition is not met then there is nothing to generate or else it gets the metadata value (itemDamage) of the operands and the mod internally performs a mathematical operation depending on the type of operator. An output BaseNumber is generated, with the result of the operation stored in the new items metadata. There are a few cases where there is nothing generated even though the operation would be mathematically correct, but our mod does not cover the use of negative numbers, fractions and decimals. We would like to implement these concepts in future versions of the extension.

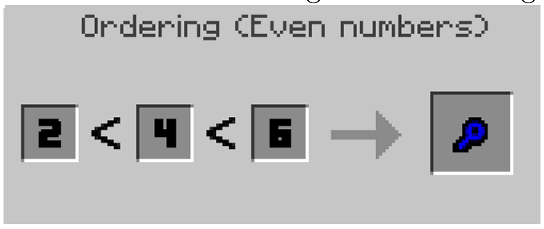
Figure 4.1: Screenshot of the Calculator's crafting matrix in use



[9]See Figure 4.4.6.

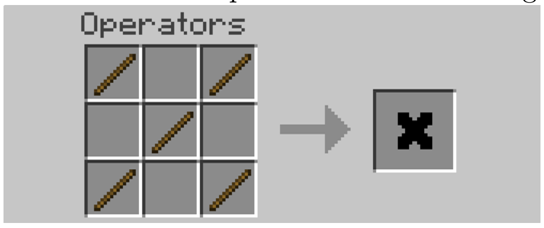
The Crafting Manager for the Ordering Bench works in a similar way, but it first tests if all the inputs are numbers and then it checks if the order and the parity of the numbers are valid depending on the type of the Ordering Table. Finally it generates a coloured key, with the colour depending on whether the bench was set to accept odd numbers, even numbers, or all numbers[10].

Figure 4.2: Screenshot of the Ordering Bench's crafting matrix in use



We decided to keep the way the core Crafting Manager works for the Operator Bench. This bench has the same behaviour as the core Crafting Table. The only modifications are the visuals and the recipes accepted. We wanted to limit the items that could be crafted by the players in the mod. We achieved this by placing only our own modified versions of the Crafting Tables in our levels. These benches accept only our custom recipes which we defined for the crafting [11] of the mathematical operators[12]. All these recipes are ShapedRecipes[13] used in the original version of Minecraft.

Figure 4.3: Screenshot of the Operator Bench's crafting matrix in use



The method signature is roughly as follows:

```java
this.addRecipe(String row1, [String row2[, String row3]]
          char itemType1, ItemStack itemStackType1[, ... char
              itemTypeN, ItemStack itemStackTypeN]);
```

A list of all the recipes added to the Operator Bench:

```java
/** A list of all the recipes added */
private List<IRecipe> recipes = new ArrayList<IRecipe>();
```

[10]See Figure 4.4.6.

[11]"Crafting is the method by which many blocks, tools, and materials are made in Minecraft. In order to craft something, players must move items from their inventory to a crafting grid. A 22 crafting grid can be accessed from the player's inventory. A 33 grid can be accessed by right-clicking a Crafting Table."

[12]See Figure 4.4.6.

[13]"Shaped recipes come in all sizes from 1x1 to 3x3. Strings are used for the recipe shape and values."[10]

```
private OperatorCraftingManager() {
    ItemStack sticks = new ItemStack(Item.stick);
    this.addRecipe(new ItemStack(EduCraft.ADD_OPR), " s ", "sss",
        " s ", 's', sticks);
    this.addRecipe(new ItemStack(EduCraft.SUB_OPR), "   ", "sss",
        "   ", 's', sticks);      this.addRecipe(new ItemStack(
       EduCraft.MUL_OPR), "s s", " s ", "s s", 's', sticks);
    this.addRecipe(new ItemStack(EduCraft.DIV_OPR), "  s", " s ",
        "s  ",'s', sticks);
}
```

4.5 Implementation of Mobs

4.5.1 General Mob Implementation Guide

All monsters(mobs) in the game are entities. Each monster we implemented was therefore also an entity and since we decided to use well known minecraft monsters the zombie and the skeleton it was natural we would just inherit from these two already existent classes. When implementing a mob there are a few simple steps. Firstly the constructor, the constructor should pass the world to the super constructor. The world should be passed into the constructor as a parameter. You can then set what item for the mob to drop by using the this keyword followed by droppedItemId and make it equal to the ID of the item you wish the mob to drop. You can then set the name of the monster which is seen by the player in game by using the method setCustomNameTag and then giving it a string.

Each monster class has many methods which help with setting which will drop. In many classes such as the skeleton class it does not use the normal method of dropping a default item. It uses the dropFewItems method so a skeleton will always drop multiple items, it may also drop a rare item through the dropRareDrop item this has a significantly lower chance. Most monsters have the rare drop method and use it, our custom monsters only implement the getDropItemId so that we can get what item the monster will drop. Not only do we not implement the other drop methods we override them and leave them empty. If theyre not overridden and left empty the monster may begin to drop other items such as helmets and bows because the minecraft engine has a chance to randomly invoke the rare drop or the case of the skeleton it will always invoke the multiple item drop method. These methods will of course be those from the superclass.

4.5.2 Discussion Of Mob Drops Event

The numberMobDropsEvent is a class which handles the dropping of number items from all custom monsters in the game. When a monster dies it calls the onEntityDrop method which has the event has a parameter. We can use this parameter to get what the monster was killed by, if we check this and it turns out to be the maths wand then we make sure we drop the number item and not some other item. This method also does one more thing it checks what monster actually died so if it was a skeleton number zombie the random number generated will be a multiple of ten if it was a number zombie it will be a number between 1 and 10 inclusive. If any other monster died the handler will do nothing and it will reach the end of the method.

4.6 Implementation of Locations

We used creative mode[14] in Minecraft to create all the different locations (levels) in our world. In order to make the visual experience in the mod more enjoyable we used a world editing tool called MCEdit to create a small forest and mountains in the surrounding area of the locations.

We have also created our own tab for easier access of custom elements in the inventory list. We added the following line to the constructors of all elements created by us:

```
setCreativeTab(EduCraft.tabEduCraft);
```

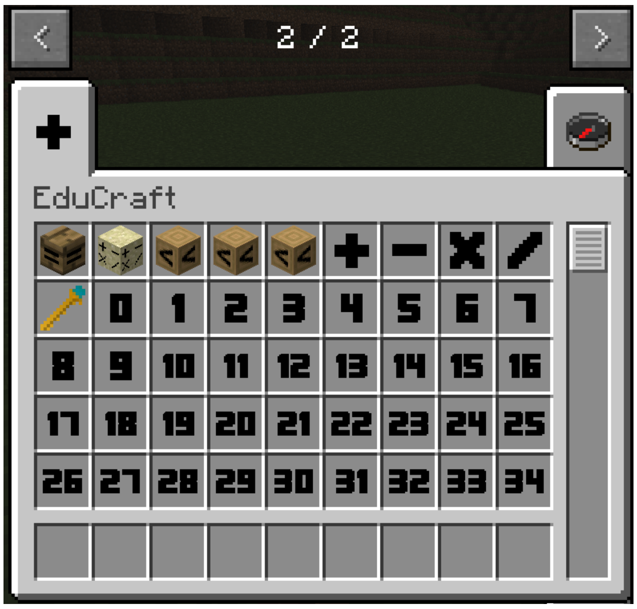


Figure 4.4: "Screenshot of EduCraft custom tab in game inventory"

4.6.1 Castle

The collaborative element of this level was implemented by setting a task to create four different operators, which would open doors to four different rooms using the hopper system in the first

[14]"Creative mode is one of the main game modes in Minecraft. This mode strips away the survival aspects of Minecraft and allows players to easily create and destroy structures. Creative mode allows players to destroy all blocks instantly (including normally-indestructible blocks such as bedrock) and the ability to fly. Players are given an infinite number of blocks to build with and no health or hunger bar thus rendering the player immune to all damage."[15]

part of the level. Each room has a key inside and all four keys are needed to allow the players to progress to the next part of this level.

In the next part each player will have to pick up a Maths Wand and kill some Zombies and collect enough numbers to be able to use the Ordering Bench. A correct ordering of numbers will generate a key that opens the door leading to the final part of this level.

In the final part the team has to obtain a specific number through a series of mathematical operations to be able to leave the level through the exit. There is a chest full of Mathematical Operators which they can use within the Calculator. The team has to kill Zombies and Skeletons to collect the numbers dropped by the monsters.

All tasks in this part are designed so no one can progress without the help of the others. This was set to be a tutorial level which would allow the players to familiarise themselves with the game play of the mod.

4.6.2 Pyramid

This level forces the team to split up at the beginning. This is achieved by only allowing two players inside the Pyramid; as soon as there are two players inside the entrance is blocked. A plan of this level's design is shown in Figure 4.6.2.

The interior of the Pyramid has two floors and it is divided into four sections. The Pyramid is surrounded by a garden with fences and walls so as to prevent the outside team from wandering around in the world. Each inner section has its own corresponding outer garden section. Neither the inside nor the outside team can go to the next section without collaboratively working towards a goal set for each section of the level.

There is an exit through the top of the Pyramid for the inside team. The inside team can only leave once each subteam has generated a key in the fourth section of the level. The outside team can climb the stairs on the side of the Pyramid in the fourth section and throw their key inside for the other team. The inside team should have two keys by this point and each key will move a block, creating a path to leave. Once all the players are reunited, they can leave through the level exit, which located in the final garden section.

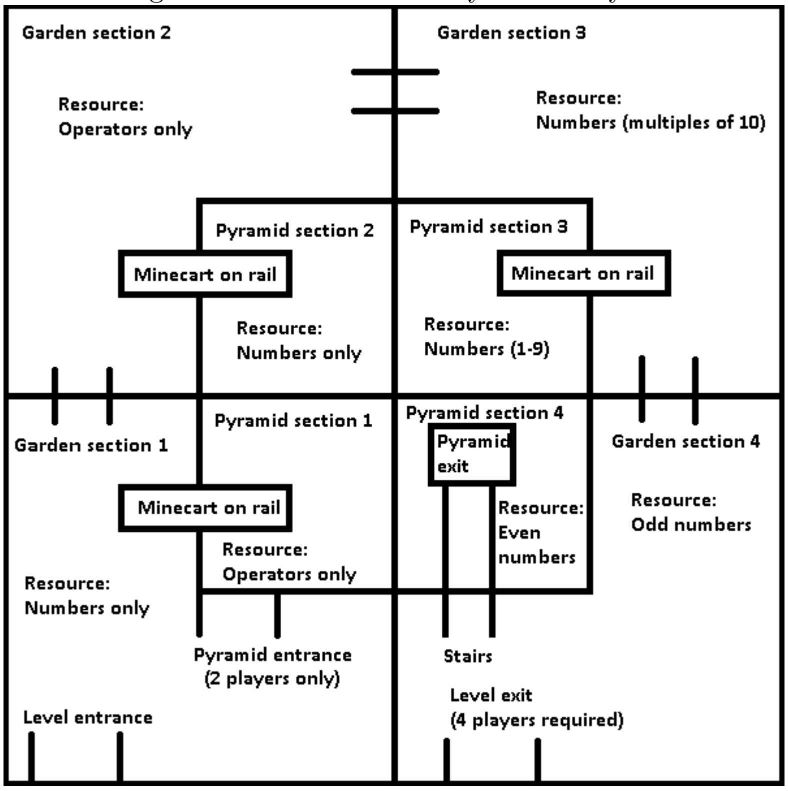
In each section the resources available are limited for each subteam, for example in one section the inside team has access to sticks and an Operator Bench, but not to any numbers. In the same section the outside team has the opportunity to kill Zombies or Skeletons to be able to collect numbers and they also have a Calculator to generate numbers.

We implemented a system that allows the teams to send items back and forth in minecarts that move on rails. This will force them to communicate their requirements to each other.

4.6.3 Rook

The Rook has two floors and on each floor there are two rooms. Only the first two players to arrive can enter each room. The subteams resources are limited in a similar manner to the Pyramid. The rooms on the same floors share a wall. In each of these walls there is a Calculator

Figure 4.5: Sketch of the Pyramid's layout



built-in, which can be used by the subteams to pass items to each other and to execute some mathematical operations together.

In order to leave the rooms the teams must generate a specific number and throw that number in the hopper by the exit. Each exit is connected to a staircase that leads to the floor above. The rooftop of the Rook is the final destination in our game. It is relatively elevated, so the players can have a look around and see the world and all the levels they have completed.

Chapter 5

Testing and Analysis

5.1 Introduction

Every project requires testing to ensure that it meets its stated objectives. This chapter describes the methods that were adopted to test the EduCraft mod 'in the field'—a description of the unit tests performed on the code itself is given in Appendix B.

5.2 Game Measures

Before describing the tests we performed, we first give a brief outline of the measures that we developed in order to give some framework for assessing the success of the game.

The measures we developed were based on the work of Nicola Whitton [22], and were gathered into three categories: engagement, collaboration, and system. These were designed to measure the children's engagement with the game, the level of collaboration they displayed, and the integrity of the system respectively. A full description of the measures is presented in Appendix C, so we omit a full discussion here.

The measures were used as guides to the observations we carried out in the following tests, in order to help focus our attention on the key things.

5.3 Pilot Study

The first test we performed was with a simple level containing a single problem. The purpose of this test was to see how quickly a child who had never played Minecraft before could learn the game and work out how to use the mod; as such, this test involved only a single player, an 11 year-old boy. He played through the level twice, with the option to ask for as much help as he needed from the observers.

5.3.1 Observations

On his first attempt, John's[1] unfamiliarity with the controls proved to be a rather large obstacle. He was uncertain how to move the character around effectively, and without knowing how Minecraft mechanics like crafting tables usually worked, he was unable to easily learn how to make the mathematical operators required. However, on the second play through he was much more adept—having learned the controls, he was more able to focus on the objectives of the level.

Despite his increased engagement with the game in general, John's engagement with the level was not quite so high. Once he had learnt the controls he was more interested in heading off to try and do his own thing rather than following the set course of the level. He missed many of the signs and messages we had set out to provide instruction for the player, and so the observers had to verbally guide him through.

5.3.2 Conclusions

We drew three key conclusions from the pilot study, the first of which provided general knowledge of how children learn to play Minecraft, whilst the last two gave particular insight into how to design levels more effectively.

Learning Minecraft

From John's initial inexperience, we determined that it would be unrealistic to expect any children who were new to Minecraft to try and learn both the basic mechanics of the game and how to use our mod at the same time. To use the mod in schools, children would either need separate time to learn how to play, or would already need to be familiar with the controls.

Attention to instructions

John's lack of attention to the signs and messages placed throughout the level was problematic. Although the use of signs to describe the level is an accepted standard amongst older players, John simply didn't read any of the messages. This knowledge was fed into our design of the final game, since we realised that to be successful we would require one of the testers to give verbal instructions to the children to guide them through[2].

Attention to route

John was also more interested in exploring the level for himself rather than following the path we had planned. This was facilitated by the very open nature of the level; there was nothing forcing

[1]All children's names have been changed.

[2]See Chapter 3 for a discussion of our approaches in game design.

John to advance as expected. This was also fed into the design of the final level—we realised we had to limit the children's options, so as to force them to follow the correct path[3].

5.4 Local Primary School

The second test was much more comprehensive, and involved a day-long visit to a local primary school. We brought the completed game[4], and had intended to have four children play through at a time in a place where we could observe their gameplay.

5.4.1 Approach

The actual day ran somewhat differently from what we had planned for. Instead of having a single group of four children to play through the game, we were introduced to the entire school, and learned that the day would involve each year group visiting various projects including ours. Instead of having the focussed attention of a group of four for the full 30–40 minutes we had planned, we had larger groups of children for a duration of approximately 10 minutes each.

The children still played through the level, with different groups taking over from each other. The two testers were still taking care to observe the groups' progress, and offered help to the children where necessary.

5.4.2 Observations

The children's engagement with the game varied dramatically from group to group and from child to child. Some paid very close attention to the game and appeared interested in completing the objectives, whilst others were more concerned with messing around and trying to break the level. Larger groups nearly always displayed lower levels of engagement, whilst the smaller groups tended to engage more. Smaller groups were also much more likely to pay attention to the instructions given by the testers, and to follow them. As with the pilot study, players who were already familiar with the game's controls engaged more than those who were unfamiliar.

There was also a lack of collaboration, shown most fully in a lack of progress through the level: no group managed to complete the 'introductory' area, and very few made it through the first area. The children were communicating with one another, but not about the task at hand. However, the groups who paid more attention to the testers did manage to achieve some level of collaboration when they were prompted to work together: one stage required the children to place three numbers in ascending order, and when the tester talked the children through the task—*"Okay, who has collected any numbers? Which numbers are they? Okay, which do you think is the smallest? Put it in the slot…"*—they did work together and talk to each other about the concepts at hand.

[3]Again, see Chapter 3.
[4]Described in Chapters 3 and 4.

5.4.3 Conclusions

The findings of the test at the primary school share some similarities with those from the pilot study, though there were also a number of additional insights we gained from assessing the children's performance on the day.

Learning environment

One of the biggest things we found was that the environment we were testing in vastly hindered our efforts to get the children to play through the level as expected— having their friends sat behind them trying to advise or take over distracted the players, and the children were generally rather excited by the change in their routine and so were distracted anyway. The large amounts of noise in the classroom, generated by the other projects, made it difficult to get a small group to focus on the instructions of the tester. As noted above, the smaller groups were able to engage and progress more, which was probably helped by the fact that there were fewer distractions.

Need for guidance

Similar to the pilot test, the players at primary school appeared to prefer to try and do their own thing rather than read the signs and follow the path of the game. This had been compensated for in the design of the level by constructing it in such a way as to prevent them from going places they weren't meant to, but they still required a lot of guidance from the testers to actually make progress.

The solution that was settled on after the first test was to have a fifth player in the game, controller by one of the testers, who would guide the other players in-game. This helped greatly, as it made it much easier for the players to see where they were meant to go, as they had someone in the game that they could follow.

5.5 Discussion

In this section, we discuss the lessons learned from the testing process according to the three categories of game measures[5], and then complete the chapter with some general remarks.

5.5.1 Engagement

Once the children learned the controls, all those who played the game were engaged quite well in one way or another. Although the children were not focussed explicitly on the learning objectives of the level, they all enjoyed playing Minecraft. We believe that the surrounding environment (i.e. the levels of noise and the number of players) was the single largest factor that impacted on engagement: those groups who experienced fewer distractions made much

[5]See Appendix C.

more progress, and it is felt that were these distractions removed entirely the players would engage much more fully.

With reference to the day of testing at the school, the children were not expecting to be learning maths—they were having an 'ICT day'—but it is felt that if the game were set in the context of a proper maths lesson, the players would be more committed to paying attention to the maths content. As it was, they were simply enjoying playing on computers.

5.5.2 Collaboration

In the test at the school, collaboration was seriously limited by the difficulty of communicating with the other players meaningfully in a noisy classroom. Our observations appear to indicate that the children required prompting to work together, rather than it being a spontaneous action, although this may be because time constraints prevented us from fully explaining the aims of the game to the children.

When prompted, the children were able to work together, and we believe that given time to test in a suitable environment where the players were not distracted by friends or noise, a greater level of collaboration would be possible.

5.5.3 General remarks

The day at the school shows that there is great potential for Minecraft to be used in education, provided that it is used in an appropriate environment. Evidence from testing would indicate that, at least to begin with, the children playing would require supervision and guidance from someone who knew how the game worked, which might limit its application in regular classes with limited numbers of teaching staff, but it is felt that given further testing EduCraft could be developed into an incredibly useful tool for teaching maths.

Chapter 6

Reflections

Reflection is a very important part of a successful software engineering project [2]. Looking back on a process you have gone through and considering what went well, what did not and what could have been done differently is a very worthwhile experience as it gives you the wisdom to improve yourself and your work in the future. Done well, reflection can give an objective evaluation of the success of a project which is unmistakably important.

6.1 Communication

Communication and group work was generally very good, although by no means perfect. We used online resources and websites to communicate and coordinate work, which was useful for keeping everyone up to date outside of our face-to-face meetings. We held formal meetings most weeks with our supervisor, Peter Blanchfield, in order to discuss progress, goals, obstacles, and to get advice on various aspects of the project. These were supplemented by informal meetings with the group alone, which helped team coordination and setting out a plan.

The importance of these meetings cannot be overstated, as they were what kept the ball rolling throughout, and ensured decisions were made, work got done in time and documentation of this all was kept so as to have a permanent record to refer to at all stages of the project. As such, these meetings were very useful, although did underline issues we had with group work; this was due to lack of involvement and communication by some members of the group at certain stages of the project. If group dynamics had been better, then we could have achieved more; for example, we had plans for an iOS app that never came to fruition due to lack of time.

Furthermore, we could have implemented more mathematical concepts to challenge older players for whom the puzzles and concepts covered in our current mod are trivial. Just as importantly were issues we had with players abusing the environment and playing the game in a non-productive way, which required level design decisions to minimise. With better group dynamics and more collective time, we could have dealt with the problems faced a lot sooner, allowing us to implement more of the desired features and meet more of the initial requirements we set.

6.2 Programming and technical issues

Progress on the mod was sometimes slowed down by technical issues, for example in our implementation of custom Minecraft items and blocks such as the numbers and the Calculator Bench. However, we dealt with these obstacles well, because when our initial ideas did not work or were implemented in a flawed manner, we would take a step back and research the problem and consider available technology and tools in order to look for workarounds. Using git for version control was crucial as we could try out some code, and if it did not solve our issue or had unexpected consequences then we could simply roll back to a previous version and start research and work on a new or modified solution.

Pairs programming was also an extremely helpful method for coding, using the combined knowledge of two people made writing competent, useful and functional code that much easier, and was immensely important in dealing with bugs, and tackling obstacles of implementation. All things considered, our approach to technical issues and the process of searching for a solution was difficult and time-consuming, but ultimately worth it as we often developed simple, clever and effective solutions.

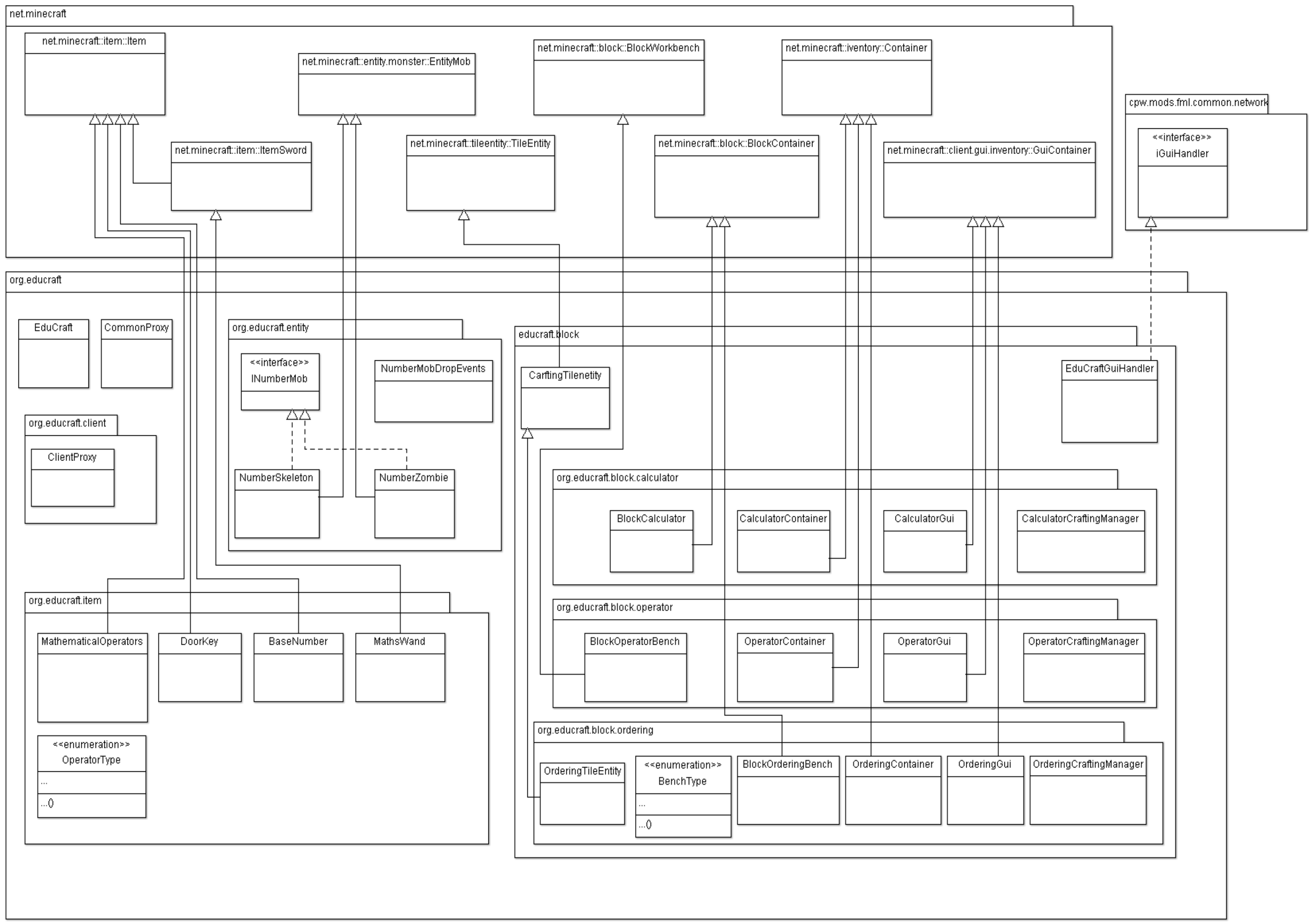
6.3 Conclusions

Overall, the project was largely a success. Despite limitations and issues faced, we have created a playable mod that fulfils the expectations of our original project description: “The project would aim at developing mods for the Minecraft game in which players could only win by collaborating with others”. Testing the game ourselves, with individual children and as well at the school showed promise that collaborative learning in Minecraft is possible, even though there were a lot of problems and shortcomings, as outlined in Chapter 5. Most of the people involved enjoyed the process and were excited about the project’s potential, and we believe that EduCraft demonstrates the usefulness Minecraft can have as an educational tool for collaborative learning, and are looking forward to further development.

Appendix A

Class Diagram

Figure A.1: UML class diagram of EduCraft



Appendix B

Unit Testing

B.1 Approach

Unit testing is considered a vital part of any software engineering project, as it allows the team to assert that the individual components of the system function as intended [21]. They also provide a convenient means of carrying out regression testing: when changes are made, the same tests can be re-run, theoretically with the same outcome.

From the outset, the EduCraft developers were concerned with how to effectively test the mod, particularly key elements of it such as the calculator block.

B.2 JUnit and Minecraft Forge

JUnit is a simple unit testing framework for the Java language [6], and as the Eclipse IDE has built-in support for JUnit, it was the first choice of testing tools to use.

However, on beginning to write unit tests for the mod, we encountered a number of problems. A number of the classes comprising the core of the Minecraft game cannot be instantiated unless there is an underlying game client running, and this includes several key classes used in our mod[1].

This limitation meant that very few meaningful test cases could be written, since the tests requiring these uninstantiable classes would not compile. It was the view of the developers that, although unit testing is considered to be best practice in software engineering [21], an alternative testing strategy would have to be devised for the project.

[1]`net.minecraft.item.ItemStack` is a prime example.

B.3 In-Game Testing

The approach decided upon was to set up a very basic ‘Test World’ with no terrain and no entities apart from the player. New blocks would be placed in this world, and their functionality tested in-game. Although such an approach means that testing documentation is not available, the entire development team agreed that this was the best solution available, given the limitations of the API.

Appendix C

Game Measures

C.1 Introduction

These measures were developed based on the PhD thesis of Nicola Whitton [22]. They were designed not to be used as interview questions, but instead as prompts to the testers, to help them focus their observations on key things.

We planned to focus our observations on three areas: measuring the children's engagement within the game, measuring the use of collaboration, and monitoring the reliability of the software while in use. Although our measures are framed as a series of questions, it was understood that interviewing children can be a difficult process, and so the measures were used as guidelines for the testers, to guide their observations.

C.1.1 Engagement

In measuring the children's engagement within the game, we wanted to know specifically how challenging they found it. While designing the game we planned to follow the optimal game design curve where as we try to the keep the difficulty increasing at a linear rate. We also looked to determine if the game concepts were easily understandable.

C.1.2 Collaboration

In measuring the use of collaboration within the game, we looked at how they communicated. We decided we didnt have time to add a chat system to the game and as such it was crucial to see how much of an impact this would have on the players ability to communicate. We also wanted to measure how effectively and easily they were able to combine operators to form the the key number they needed to progress.

C.1.3 System

Lastly, we wanted to see how the software we had developed measured up under actual use, to help us test the system and find areas for further development. It would also complement the unit tests we have written for the software, to provide a fuller picture of the software's reliability.

C.2 Measures

C.2.1 Engagement with the game

1. Measure average time it takes for a group of players to complete a level
2. Ask general questions about how challenging and immersive the game was
    (a) Were there any points where they were confused or not sure what was going on?
    (b) Were there aspects of the game that were too easy or too hard?
    (c) How did the difficulty of the game change as you progressed through levels
3. How much of game time is spent on task?
    (a) What possible distractions are there?
4. How often do they ask for help understanding the game concept?
    (a) What sort of questions did they ask?
5. Did they like the zombie characters?
6. What were their thoughts on the level maps?
    (a) Were they interesting and varied?

C.2.2 Use of collaboration

1. How well did they communicate?
2. Did they understand the need to collaborate in order to advance?
3. Did they use their maths skills or did they randomly guess and match operators?

C.2.3 System integrity

1. Did the game crash?
    (a) What was the reason?

Appendix D

User Guide

This appendix presents a brief guide to using EduCraft. More extensive documentation, including videos, is available at `TosinAF.github.io/EduCraft`. URLs to the relevant videos are included in the sections below.

D.1 Controls

EduCraft uses the same controls and follows the same basic principles of regular Minecraft, and so anyone with a basic understanding of Minecraft should be able to play Educraft. Even without experience, the controls and concepts can be picked up quickly, and the main level design we have produced incorporates a tutorial section at the start.

A full description of Minecraft's default controls is available on the official Wiki page [18].

D.2 Tutorial walkthrough

This section provides a summary walkthrough of the first level of the game, which is used as an introduction to EduCraft's new mechanics.

D.2.1 First room

The first step in the tutorial is to collect logs, which can be turned into planks and then sticks for crafting purposes[1]. Sticks are important as they are what the player needs to craft operators. This is done by creating the pattern of the desired operator in the Operator Bench, which can be used in a similar way to a crafting table in regular Minecraft. Players need to make patterns +, -, /, or X for plus, minus, divide, and times [2]. The final step is to unlock doors by throwing the appropriate operator into the hopper for the adjacent door[3], and then throwing the keys

[1]Turning logs into sticks: `www.youtube.com/watch?v=Q3aMJ5jwxuM`

[2]Creating operators: `www.youtube.com/watch?v=Mrg_mkqP-D8`

[3]Throwing items into hoppers: `www.youtube.com/watch?v=0a-KPj_FB80`

found inside into the hoppers at the end of the corridor.

D.2.2 Second room

The second room introduces players to the concept of killing enemy characters in order to collect numbers[4]. The second room requires the player to collect three numbers and then place them in an Ordering Bench, in order to obtain a key to unlock the second door[5]. The ordering bench in the second room will accept all numbers; benches later in the game will accept either odd or even numbers only.

D.2.3 Third room

The final room works in a similar fashion to the second, except that the players must use the Calculator provided to produce a target number [6], rather than using and ordering bench to gain a key.

D.3 Collaboration

Educraft requires collaboration between teams of players in order for progress to be made. One way this is done is by the necessity for players to pass resources (operators, numbers) to each other by using rail carts. The player places the item(s) they want to send to the other team into the chest on the cart, and then presses the button to send the cart [7].

[4]Killing monsters: `https://www.youtube.com/watch?v=ZswqlwMR3f4`

[5]Using the ordering bench: `https://www.youtube.com/watch?v=MQTVB4etTfE`

[6]Using the calculator: `https://www.youtube.com/watch?v=uxbQ4CySm30`

[7]Using a minecart to exchange items: `https://www.youtube.com/watch?v=k-GD8Vr8fhc`

Bibliography

[1] J. Brand and S. Kinash, "Crafting minds in minecraft," *Learning and Teaching papers*, vol. 53, 2013.

[2] J. Dowling, T. Schafer, V. Cahill, P. Haraszti, and B. Redmond, "Using reflection to support dynamic adaptation of system software: A case study driven evaluation," *Lecture Notes in Computer Science*, vol. 1826, pp. 169–188, 2000.

[3] M. P. J. Habgood, "The effective integration of digital games and learning content," Ph.D. dissertation, University of Nottingham, July 2007.

[4] D. Jacques and G. Salmon, *Learning in Groups*, 4th ed. Abingdon, England: Routledge, 2000.

[5] D. W. Johnson and R. T. Johnson, *Learning Together and Alone*, 4th ed. Needham Heights: Allyn and Bacon, 1994.

[6] JUnit, "Junit." [Online]. Available: junit.org

[7] Minecraft Forge Wiki, "Base Mod Class - Basic Modding - Minecraft Forge Wiki," March 2014. [Online]. Available: http://www.minecraftforge.net/wiki/Basic_Modding#Base_Mod_Class

[8] ——, "Creating a Container - Containers and GUIs - Minecraft Forge Wiki," March 2014. [Online]. Available: http://www.minecraftforge.net/wiki/Containers_and_GUIs#Creating_the _container

[9] ——, "Proxy Classes - Basic Modding - Minecraft Forge Wiki," March 2014. [Online]. Available: http://www.minecraftforge.net/wiki/Basic_Modding#Proxy_Classes

[10] ——, "Shaped Crafting - CCrafting and Smelting - Minecraft Forge Wiki," March 2014. [Online]. Available: http://www.minecraftforge.net/wiki/Crafting_and_Smelting#Shaped_Crafting

[11] Minecraft Wiki, "Minecraft in education - Minecraft Wiki," December 2013. [Online]. Available: http://minecraft.gamepedia.com/Minecraft_in_education

[12] ——, "Adventure - Minecraft Wiki," March 2014. [Online]. Available: http://minecraft.gamepedia.com/Adventure

[13] ——, “Blocks - Minecraft Wiki,” March 2014. [Online]. Available: http://minecraft.gamepedia.com/Blocks

[14] ——, “Crafting Table - Minecraft Wiki,” March 2014. [Online]. Available: http://minecraft.gamepedia.com/Crafting_bench

[15] ——, “Creative - Minecraft Wiki,” March 2014. [Online]. Available: http://minecraft.gamepedia.com/Creative

[16] ——, “Item - Minecraft Wiki,” March 2014. [Online]. Available: http://minecraft.gamepedia.com/Item

[17] ——, “Mobs - Minecraft Wiki,” March 2014. [Online]. Available: http://minecraft.gamepedia.com/Mobs

[18] MinecraftWiki, “Controls - Minecraft Wiki,” March 2014. [Online]. Available: http://minecraft.gamepedia.com/Controls

[19] Mojang AB, “Minecraft,” November 2013. [Online]. Available: http://www.minecraft.net

[20] D. Short, “Teaching scientific concepts using a virtual world - minecraft,” *Teaching Science*, vol. 58, no. 3, 2012.

[21] I. Sommerville, *Software Engineering*, 9th ed. London, England: Pearson Education, 2011.

[22] N. J. Whitton, “An investigation into the potential of collaborative computer game-based learning in higher education,” Ph.D. dissertation, Edinburgh Napier University, August 2007.

[23] E. Yackel, P. Cobb, and T. Wood, “Small-group interactions as a source of learning opportunities in second-grade mathematics,” *Journal for Research in Mathematics Education*, vol. 22, pp. 390–408, 1991.