Department of Computer Science

Jamie Birch, Sarah Carter, Ching Chi Chan,

Matthew Ellams, Sam Grant, Alexander Lorimer

\_\_\_REPORT\_\_\_

**CODE a**

**COLONY**

MS50 Computer Science Conversion,

Software Development,

Group **B**ees

2016

**Declaration of Mark Distributions**

We hereby declare that the following distribution of marks have been agreed by all of the undersigned, and that the work detailed in this document is the sole product of the group members.

Jamie Birch % \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Sarah Carter % \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Ching Chi Chan % \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Matthew Ellams % \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Sam Sutherland Grant % \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Alex Lorimer % \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

# Contents

[Contents 3](#_Toc441338741)

[**Abstract** 6](#_Toc441338742)

[**Introduction** **1.0** 7](#_Toc441338743)

[Computer Science Education 1.1 7](#_Toc441338744)

[Gender Gap 1.2 8](#_Toc441338745)

[Industry Stakeholders 1.3 9](#_Toc441338746)

[Target Audience 1.4 10](#_Toc441338747)

[Bristol Grammar School, IT Class Focus Group (Pre-development) 1.4.1 10](#_Toc441338748)

[Bristol Grammar School, meeting with the Head of IT (Pre-development) 1.4.2 12](#_Toc441338749)

[Aims and Objectives 1.5 13](#_Toc441338750)

[**Concept Development** **2.0** 14](#_Toc441338751)

[Initial Concepts 2.1 14](#_Toc441338752)

[Text Adventure Game 2.1.1 14](#_Toc441338753)

[CPU Puzzle Game 2.1.2 15](#_Toc441338754)

[Bee Concept 2.1.3 17](#_Toc441338755)

[Post Focus-group Concepts 2.2 18](#_Toc441338756)

[CPU Mario 2.2.1 18](#_Toc441338757)

[Shrek Themed Game 2.2.2 19](#_Toc441338758)

[Bee Concept (further developed) 2.2.3 20](#_Toc441338759)

[Background Research 2.3 21](#_Toc441338760)

[Relevance and Scope 2.3.1 21](#_Toc441338761)

[Currently on the Market 2.3.2 23](#_Toc441338762)

[Scratch 2.3.2a 23](#_Toc441338763)

[Educational Coding Games 2.3.2b 24](#_Toc441338764)

[Swarm Manipulation 2.3.2c 24](#_Toc441338765)

[Final Concept 2.4 26](#_Toc441338766)

[Unique Selling Points 2.4.1 26](#_Toc441338767)

[**Requirement Specification 3.0** 27](#_Toc441338768)

[Introduction 3.1 27](#_Toc441338769)

[Overview 3.2 27](#_Toc441338770)

[Product Components 3.3 27](#_Toc441338771)

[Opening title screen 3.3.1 27](#_Toc441338772)

[Audio 3.3.2 27](#_Toc441338773)

[HUD 3.3.3 27](#_Toc441338774)

[World and terrain 3.3.4 28](#_Toc441338775)

[Characters 3.3.5 28](#_Toc441338776)

[Timed Events 3.3.6 28](#_Toc441338777)

[Interactive objects 3.3.7 28](#_Toc441338778)

[Use Characteristics 3.4 28](#_Toc441338779)

[Dependencies 3.5 29](#_Toc441338780)

[Constraints 3.6 29](#_Toc441338781)

[Time 3.6.1 29](#_Toc441338782)

[Skillset 3.6.2 29](#_Toc441338783)

[Fair Use and Copyright 3.6.3 29](#_Toc441338784)

[Divided Focus 3.6.4 30](#_Toc441338785)

[Quality Assurance 3.7 30](#_Toc441338786)

[Suitability 3.7.1 30](#_Toc441338787)

[Efficiency 3.7.2 30](#_Toc441338788)

[Flexibility 3.7.3 30](#_Toc441338789)

[Security 3.7.4 30](#_Toc441338790)

[Maintainability 3.7.5 30](#_Toc441338791)

[Portability 3.7.6 31](#_Toc441338792)

[Stability 3.7.7 31](#_Toc441338793)

[Usability 3.7.8 31](#_Toc441338794)

[Use Cases 3.8 31](#_Toc441338795)

[**Design Implementation 4.0** 34](#_Toc441338796)

[Module Design 4.1 34](#_Toc441338797)

[Systems Architecture 4.2 34](#_Toc441338798)

[Graphics 4.3 34](#_Toc441338799)

[Opening title screen 4.3.1 34](#_Toc441338800)

[Game world 4.3.2 34](#_Toc441338801)

[Game objects 4.3.3 34](#_Toc441338802)

[User interface 4.3.4 34](#_Toc441338803)

[Audio Design 4.5 35](#_Toc441338804)

[Core Features 4.6 36](#_Toc441338805)

[Testing 4.7 36](#_Toc441338806)

[Tools 4.8 37](#_Toc441338807)

[Collaboration 4.8.1 37](#_Toc441338808)

[C 4.8.2 37](#_Toc441338809)

[Media 4.8.3 38](#_Toc441338810)

[Version control 4.8.4 38](#_Toc441338811)

**Abstract**

Code A Colony is a game that gives players significant control over a colony of bees. This colony can exhibit complex, collective behaviors determined by player-constructed algorithms. Targeted at Key Stage 3 level students, the aim is to develop and reinforce computing skills that have become a statutory part of the UK curriculum as of September 2014 (with a significant focus on the application of logic and algorithmic problem solving).

We have designed the game to allow players to experiment with custom problem solving/swarm intelligence algorithms, within a graphically rich, interactive context. The bees have an intuitive API, and are endowed with a small amount of memory along with an ability to conditionally interact with both their local environment and with neighboring bees. The controlling algorithm is constructed and manipulated by the player graphically, represented as visual connections between selected nodes (which represent functions) within a graph.

This report details the research, development and management that went into the Code A Colony project, while also documenting our experiences of the agile software development processes we employed.

**Introduction 1.0**

Technology has changed the way in which we live our everyday lives, from communicating with friends all over the world with a simple click, to the way in which we find a new recipe, or even how we navigate from A – Z. Computers have emerged as ubiquitous devices; increasingly powerful, portable, and intuitively interactive. But this apparent simplicity belies their growing complexity and sophistication. Few of us are capable of seeing past the opacity of modern computers, to reason about their underlying technologies, despite the vast majority of the population relying on digital technology daily.

In the 1980s - at the very cusp of the personal computer revolution - a gap in the computer literacy of the public had been noticed. This provoked the BBC’s Computer Literacy Project and the launch of the BBC Micro (a personal computer designed for computing education). As described by the project editor David Allen, *“The aim was to democratise computing. We didn’t want people to be controlled by it, but to control it.”*[[1]](#footnote-2) This initiative introduced millions of children to the field of computing, and inspired a generation of computer scientists and engineers.[[2]](#footnote-3)

Yet growing concerns still exist, revolving around a disparity between computing education and the demands of the industry. In 2011 Eric Schmidt (then Chairman at Google) criticised Britain’s education system, stating *“your IT curriculum focuses on teaching how to use software, but gives no insight into how it is made.”*[[3]](#footnote-4) Inspired by the success of the BBC Micro, and fueled by the current discourse surrounding computer science education, this report details our attempt at designing and implementing a system to educate and inspire Key Stage 3 level school children about digital technology. The intention is to help demystify core computing concepts to a new audience, reinforcing recent statutory changes to the national curriculum regarding computer science education.

## Computer Science Education 1.1

In September 2013 the UK Department for Education published a document explaining significant changes to the national curriculum, with a legislative move (at Key Stage 3 level, and earlier) away from general “ICT” and the use of common software, towards teaching students to *"apply the fundamental principles and concepts of computer science, including abstraction, logic, algorithms and data representation."[[4]](#footnote-5)* A central goal of this reform is to ensure that all pupils, *“can analyse problems in computational terms, and have repeated practical experience of writing computer programs in order to solve such problems.”*[[5]](#footnote-6)

Before these changes came into effect in September 2014, concerns about a lack of appropriately skilled teachers prompted millions in investment (from both public and private sources) towards computer science training programmes for secondary school teachers.[[6]](#footnote-7) Never-the-less, in December 2015 the chief of the British Computing Society (appointed by the government to lead these reforms) stated that teachers are still struggling with the new curriculum, and that it could be another five years until it will be successfully taught[[7]](#footnote-8) across the country.

A significant problem appears to be that not all schools are fortunate enough to have access to specialist, appropriately qualified staff. Government investment continues to be aimed at increasing the number of “master” computing teachers, but these targets continue to be missed.[[8]](#footnote-9) Additionally, with roughly 60% of parents not aware of the new curriculum’s introduction in 2014,[[9]](#footnote-10) support for computer science appears to be limited both as school and at home. Additionally the ratio of girls to boys who decide to study computer science at GCSE level is shockingly low.

## Gender Gap 1.2

Computer science as a subject at GCSE level in 2014 was significantly less popular than ICT. This may have been influenced by initial implementation problems. Nevertheless, the average ratio of girls to boys in computer science cohorts was significantly less than in ICT (see graphs below). Despite this, as part of further reforms, the government announced plans in November 2015 to scrap GCSE and A-Level ICT altogether. The intention is to cast more focus on computer science. As stated in the original document, ministers have “taken the decision not to approve two GCSEs and A levels in a similar qualification space. The IT GCSE and IT A level will not be redeveloped.”

*Statistics collated Joint Council for Qualifications ‘Results 2015’ http://www.jcq.org.uk/Download/media-centre/news-releases/gcse-press-notice---wales*

This has sparked significant debate,[[10]](#footnote-11) largely hinging on concerns about further alienation and narrowing of the curriculum for girls.

*Statistics collated Joint Council for Qualifications ‘Results 2015’ http://www.jcq.org.uk/Download/media-centre/news-releases/gcse-press-notice---wales*

*Statistics collated Joint Council for Qualifications ‘Results 2015’ http://www.jcq.org.uk/Download/media-centre/news-releases/gcse-press-notice---wales*

## Industry Stakeholders 1.3

The recent computing focused curriculum reforms were instigated largely by pressure from tech industry leaders,[[11]](#footnote-12) asserting that significant change will be essential if the UK is to remain globally competitive in certain computer-focused disciplines. Next Gen, for example, is a report published in 2011, strongly recommending reforms the UK curriculum to bring computer science in as an essential discipline.[[12]](#footnote-13) The Next Gen report was responding to declining national competitiveness in the computer game and visual effects industries, along with concerns about increasingly “having to source talent from overseas because of skills shortages at home.”[[13]](#footnote-14)

The report also strongly recommended the appointment of “the best teachers to teach computer science”, and the use of gaming and visual effects within classrooms, suggesting, “Schools harness the ‘cool factor’ and technology potential of video games and visual effects to draw in more young people to computer science.”[[14]](#footnote-15) It is too early to see the industry ramifications of the reforms to the national curriculum, but so far the indications are that the classroom learning experience has not been enough to effectively draw children in, particularly the girls.

## Target Audience 1.4

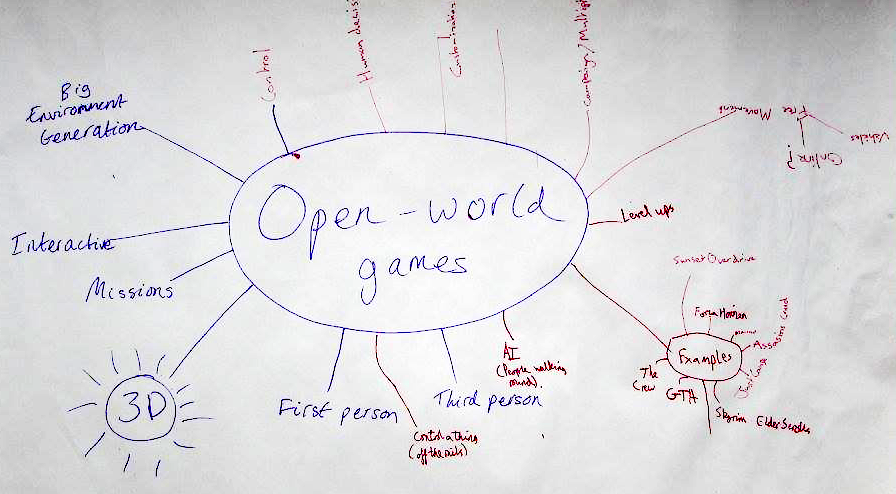
While secondary market research has acted to inform our motivations for this project, our efforts in the development of Code A Colony have been guided significantly by primary market research. We have tried (where possible) to engage directly with Key Stage 3 educators as well as a user sample of our target audience (year 9 students of Bristol Grammar School, aged 13–14, both male and female). This has acted as an invaluable source of information, while allowing us to practice the agile programming concept of short feedback cycles.

### Bristol Grammar School, IT Class Focus Group (Pre-development) 1.4.1

An initial visit to the school took place on Thursday 8th October, in which we were granted an hour for a focus group with year 9 pupils. For this we devised a series of activities designed to acquire information about IT education, along with opinions and preferences regarding computer games (as well as other topics).

During the first 5 minutes we introduced ourselves and engaged in a class discussion about what the pupils had been doing and learning so far in terms of computer programming. The answers **ranged significantly**, from having little experience at all, to actively engaging in interesting web-scripting projects. We were also introduced to their use of Scratch, **a visual programming language**. This came acted as a great source of inspiration for our project.





We then divided the class into four lines and set up a relay game where the first person in each line ran up to the front of the class to write an interest on an A2 sheet of paper. After several minutes the group with the longest list would win. We then handed out coloured stickers to each pupil and asked them to place a sticker next to the three interests that they prefer (on any amongst the 3 other teams' sheets of paper). This allowed us to get a ranking of the most common interests, while sparking class discussion.

We subsequently organised them into smaller groups, asking each to pick an interest and brainstorm possible connections to the field of computer science. Finally, we asked the groups to design a game relating to this interest, keeping in mind that it must include an educational computer science element. Pupils were encouraged to draw mind maps, and these materials can be found in Appendix 1. The results from these activities are listed below.

Class Discussion & Groups Concepts

* Emphasis on interactive, **explorable**, non-static, **‘open worlds’**
* Connection to **cartoons**, and perhaps recognisable characters
* ‘Mario’ concept , where a player **collects blocks of code** and has to figure out what order they should go in before handing to the CPU
* Importance of **story**-based progression

Pen Relay Game

Highest Ranking Interests

* South Park (4)
* Xbox (4)
* Hockey (3)
* Family Guy (2)
* Football (2)
* FPS CoD / cs:go (2)
* GTA-game (2)
* Open world games (2)
* Rugby (2)
* Antelopes (1)
* Caramel (1)
* Phone (1)
* Shrek (1)
* Social Media (1)
* Strategy (1)

Cartoons

Digital Technology

Sport

Commonalities

In addition to these activities, we also asked pupils to fill out a questionnaire at the end of the class, ranking twelve game aspects in terms of importance to their experience (see pie chart below; ranks on next page).

Game Aspect Questionnaire (out of 10 – lowers scores rank most highly)

| **Category** | **Combined score** |  | **Rank** |
| --- | --- | --- | --- |
| Graphics | 38 |  | 1 |
| Story | 43 |  | 2 |
| Competing with others | 61 |  | 3 |
| Constant interaction | 68 |  | 4 |
| Innovation | 75 |  | 5 |
| Characters | 76 |  | 6 |
| Cooperation with others | 77 |  | 7 |
| Single player gameplay | 91 |  | 8 |
| Checkpoints | 94 |  | 9 |
| Levelling up | 96 |  | 10 |
| Sound design | 105 |  | 11 |
| Replayability | 115 |  | 12 |

The main insights we gained and took forward from these activities were that:

* **Cartoons and comedy** seem to be a useful tool to make game ideas more appealing. Game characters that are caricatured, and themes/challenges that contain an element of humor could be useful devices to draw in our target audience.
* **Story- / stage-based progression** (“levelling up”) was also a very popular concept. This could in some way be tied to an aspect of humour, but also would be very useful in the form of a progressive tutorial.
* **Game environments** that can be **openly navigated**, and freely explored and interacted with are important.
* A sense of **collecting items, points, or “code blocks”** could act as the main challenge, allowing for some element of competition (competing against your previous score, or the score of a friend).

### Bristol Grammar School, meeting with the Head of IT (Pre-development) 1.4.2

On top of our detailed discussions with school pupils, we were also granted a 30 minute meeting with Amy Finney, the head of IT at Bristol Grammar School. Her very informative insights regarding the development of a computer science education game can be summarised in three key points.

The game should be:

* **FORGIVING**/robust: “If it is too difficult, and too easy to make mistakes, they will lose interest”. (Block-based scripting like Scratch is very good for this!).
* It should be **RESPONSIVE**, (i.e. have instant results): “If the children don’t see the results of what they’re doing very quickly, then they will lose interest”. That is to say, we should avoid having long periods where users have to work on code with nothing else happening.
* It should have **DEPTH:** “Users will have all sorts of different skill levels – try to appeal to more than just the most experienced users”. We should thus ensure that the game avoids becoming one-dimensional and has variety in the challenges.

The teacher also took us through an introduction to Scratch, which will be detailed in section “…“ (Currently on the Market). She emphasised the range of different ability and skill levels of pupils within the school, in terms of programming. She also mentioned a large disparity between schools, in terms of the amount of focus given to programming and contemporary computer science education.

## Aims and Objectives 1.5

Code A Colony is intended to reinforce and supplement computer science topics that have become a fundamental part of the national curriculum at Key Stage 3. Affecting real change means reaching homes as well as schools, and for this reason, we have focused our efforts purely on software (as opposed to costly hardware) for the creation of an engaging, educational game. Guided by our research findings, we have isolated a series of aims:

* To develop gender-neutral concepts (in terms of theme and graphics) that should appeal to girls, as much as boys (applying elements of humour and cartoon style graphics as a source of common interest).
* To establish a framework into which progressive stories, challenges and tutorials could be easily added and extend.
* To produce a visually rich, responsive game, with sufficient depth to be enjoyed by children with more coding experience, while being accessible (and forgiving in nature) to absolute beginners.

Our overall objective is to demystify computer science to Key Stage 3 children, and to empower them to *“analyse problems in computational terms”*[[15]](#footnote-16) through building a simple understanding of algorithms and Boolean logic (key objectives as defined by the recent UK curriculum reforms). A secondary objective is to finish a well-polished, playable prototype of this game by the necessary deadline (22/01/26).

**Concept Development 2.0**

## Initial Concepts 2.1

We began to develop concepts for this project while gathering secondary research on the state of computer science education. The three that are briefly present in this section (Text Adventure Game, CPU Puzzle Game and Bee Game) were simple concepts proposed before our focus group at the Bristol Grammar School. We have applied SWOT analysis in light of our focus group discussions. In section 2.2 (Post Focus-group Concepts) we will show how this allowed the ideas to morph and develop significantly, with some sub-concepts even proposed directly by the pupils.

### Text Adventure Game 2.1.1

An early concept proposed was a text based adventure game, with the idea that many different scenarios and themes would be easy to produce after establishing a general framework. The idea was to introduce players to programming in a context of their choice; being able to control a Rock Star on tour, a soldier in a zombie apocalypse or a wizard fighting enemy trolls. Players would control these characters in the context of a story, making decisions for that character using general programming syntax and terminology (employing functions that take variable arguments). As an example, a character might be moved left or right, or instructed to defend itself against a specific enemy.

The zombie wants to eat your brains… what do you do?

setSpeed(MAX\_SPEED);

run(LEFT\_

API

The decisions made by the player would allow them to either progress through the game, reaching harder challenges, or move them back in the game to less sophisticated problems. In this way the game would accommodate both beginners and children with more experience.

|  |  |
| --- | --- |
| **Strengths**   * Story element highlighted as important by target audience * Easily extensible and customisable themes, with potential to target both boys and girls * Could be developed to teach the application of a wide range of computing concepts * Could be developed to allow players to create their own story lines and challenges. | **Weaknesses**   * Difficult to implement enough graphics to enhance the game and support many different themes and scenarios * Lack of competitive element (or concept of score) important to the target audience * Unlikely to spark enthusiasm for the target audience |
| **Opportunities**   * Role playing games have been hugely successful in the past (Monkey Island, Dungeons & Dragons) | Threats   * **Concept** is unoriginal and outdated compared to modern games that are highly accessible |

### CPU Puzzle Game 2.1.2

The concept attempted to flip the relationship between programmer and computer around, as a novel approach to this project. The idea is that program instructions would be displayed to the player, who would need to act analogously to a Central Processing Unit (and its output and storage peripherals) to interpret these instructions, and perform the correct actions to complete tasks. An example would be to move objects into instructed positions, or colour objects correctly, etc.

As the game progresses, in order to increase difficulty, lower level instructions could be used to reduce the level of abstraction and proximity to human language. At the lowest levels of abstraction (/highest levels in the game), an ability to interpret hex and/or binary would be required to perform actions correctly.

int x = 10;

int y = 6;

moveSquareTo(bigSquare, x, y \* 2);

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 0.. | 2.. | 3.. | 4.. | 5.. | 6.. | 7.. | 8.. | 9.. | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 |  |  |  |  |  |  |  |  |  |
| 0 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 1 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 2 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 3 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 4 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 5 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 6 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 7 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 8 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 9 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 10 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 11 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 12 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 13 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |

|  |  |
| --- | --- |
| **Strengths**   * An interesting/original game concept, flipping the human/computer relationship around * Player is able to learn about lower level computing concepts, and computer architecture, in addition to higher level programming syntax | **Weaknesses**   * Novelty may wear off quickly * Relatively static puzzles and repetitive context may become boring * No interactive explorable world, though deemed as important to our focus group * Doesn’t allow players to become familiar with the concept of writing programs themselves |
| Opportunities   * Teaching children more about the complex interaction between the hardware components inside computers fills a gap in the new curriculum (more focused on software and data representation) | Threats   * There are many, many puzzle and brain teaser games on the market. It may be **difficult to compete** without a better source of differentiation |

### Bee Concept 2.1.3

This concept was the earliest form of our Code A Colony game, and was sparked simply from a discussion about our team name, The Bees (Group B). The concept is that a player would be able to program instructions into a colony of bees, telling them where to find resources and what to do with them (such as bringing them back to the hive to earn points). With a large number of bees carrying out player defined instructions, interesting patterns might emerge, resulting in a game that is interesting to watch as well as to directly control.

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |

setBeeDest(bee[4], 30, 9);

harvestFlo

Rather than significantly morphing into a new idea, this concept has only been added to since our initial focus group at Bristol Grammar School. A SWOT analysis for this idea, in its further developed form, will therefore be presented in the following section (2.2 Post Focus-group Concepts)

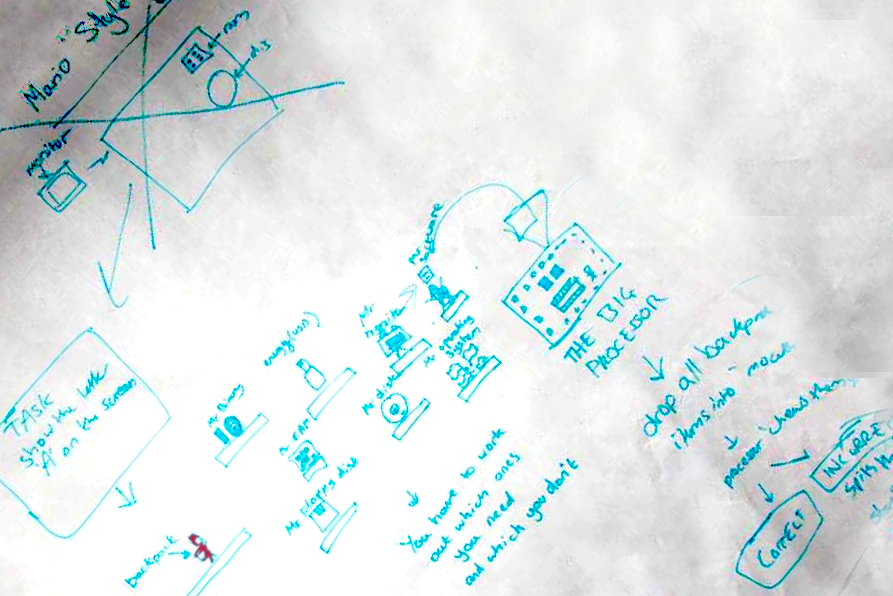
## Post Focus-group Concepts 2.2

Towards the end of our initial focus group at Bristol Grammar School, we each walked around discussing elements of our initial concepts with groups of pupils (in addition to the ideas that they had been brainstorming). Some groups were able to combine these initial concepts with their own ideas, resulting in interesting game proposals that proved popular with the rest of the class.

Additionally, before leaving we presented the basic idea of our bee concept. This was met enthusiastically, with pupils mentioning how they think the idea could be improved. The following three concepts are the result of this interaction with our focus group.

### CPU Mario 2.2.1

CPU Mario is an idea largely proposed by the pupils, in which a player would explore a “Mario” style world, but instead of collecting coins they would collect different parts of a program or instruction. Once the full instruction has been collected, the player would make their way to the CPU, placing the code fragments in the correct order and performing appropriate actions to progress to the next level. This concept incorporates elements of our early CPU Puzzle Game idea, but also extends it to include an **explorable world** with game characters. The idea proved to be very popular with the majority of the class, both girls and boys (having been proposed by a group of girls).



|  |  |
| --- | --- |
| **Strengths**   * Interesting twist on an otherwise familiar and popular format would make this easy for new-comers to engage with * Does not put the educational subject matter at the forefront. Fun game mechanics would draw players in * Difficulty levels could be easily manipulated to suit differing abilities without altering the nature of the game | **Weaknesses**   * Platforming concept is slightly passé, 30 years on from its popularization * It might difficult to differentiate our product in a sea of Mario clones * Would require massive effort in level design to give the necessary game depth, which might detract from the more fundamental programming element of the project |
| **Opportunities**   * Primary research indicates excitement about the concept from our sample group, which will be will be key to captivating our target audience * Could allow for player designed levels, potentially reducing our overall work load | Threats   * **Copyright issues** (if spun-off directly from Mario) * Player designed levels would require the design of a user-friendly level editor, and **requires initial time investment** from early adopters which isn’t guaranteed |

### Shrek Themed Game 2.2.2

This is another idea that was proposed largely by the pupils, but which also represents progression from our earlier text adventure game (by the inclusion of graphics and individual mini-games/missions). Our sample group voted Shrek as one of their top interests, and some of the students put forth a Shrek themed mini game series. The players would control Shrek’s actions using an appropriate API. The entire story of the game would be subdivide into separate missions and mini-game tasks that exercise a range of programming skills and concepts.



|  |  |
| --- | --- |
| **Strengths**   * The character featured in the game ranked highly amongst our target audience * Story element highlighted as important by target audience | **Weaknesses**   * Time constrictions would make it difficult to produce a varied mini game series effectively |
| **Opportunities**   * Varied mini games allow us to appeal to the interests and learning techniques of a range of students (if they do not learn from one game, they may relate to the format of another) | Threats   * **Copyright issues** (if spun-off directly from Shrek) |

### Bee Concept (further developed) 2.2.3

During our discussion of the Bee concept at the end of the focus group, it became apparent that we could incorporate many elements of the pupils’ ideas into this concept, while effectively responding to the advice of the Head of IT at the school, in terms of game depth and responsiveness. Giving players control over a large colony, where simple instructions will have wide ramifications, means that the player will be able to identify the results of their actions quickly. With a clever arrangement of instructions, sophisticate swarm behaviors could be achieved, giving enough depth to allow more experienced individuals to explore and have fun.

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |

stingPicnicPerson(32, 12, NO\_MERCY);

collectFood();

Additionally, the idea of collecting blocks of code and arranging them in the correct order is something that would be easy to incorporate. This would also allow us to move away from a text-based editor, to more of a Stratch related, graphical programming system (see Currently on the Market 2.3.2). Mini-games could be included to the extent that a variety of challenges could be added, while also being presented in the context of a progressive story. Humourus challenges, such as stinging people to steal their food, could make the game more appealing (this has been indicated by our research findings).

|  |  |
| --- | --- |
| **Strengths**   * Capacity for a high degree of game play depth and responsiveness * Potential for complex colony behaviours provides more scope for an interesting and flexible programming interface * Casts more attention on the players developing interesting control algorithms, supporting aim of national curriculum | **Weaknesses**   * Time constrictions may become an issue if we attempt to incorporate mini-game challenges and an element of story |
| **Opportunities**   * Story and mini-game/challenge elements proposed in other ideas can easily be included | Threats   * Players may find it difficult to establish effective swarm behaviours and could **lose** interest if too difficult |

## Background Research 2.3

Given the positive response this concept received at the school, and the potential for incorporating concepts and ideas from the pupils, the programmable bee concept became our top choice and the focus of further research and development.

### Relevance and Scope 2.3.1

Algorithms are a key feature of the reformed national curriculum at Key Stage 3, and our bee colony concept is uniquely placed to explore this topic. Autonomous problem solving/learning algorithms control and influence many aspects of society, from weather forecasting to stock trading activity,[[16]](#footnote-17) or from transport planning and scheduling to date matching[[17]](#footnote-18) (the list is ever growing). But despite the complexity of the problems such algorithms tackle, frequently the approach that is employed can be defined by simple rules that are far easier to understand. These problem solving algorithms are often inspired or informed by the natural world, as we learn from the processes and behaviours that biological systems have evolved over millions of year to overcome complex challenges.

Social insects are a common area of study for this purpose. As a simple example, a swarm intelligence algorithm might determine optimally short paths between multiple points by replicating ant colony behavior. Ants tackle such problems by individually leaving a trail of pheromones, signaling whether or not they have recently found food. Other ants can randomly explore, but are inclined to follow attractive trails. Over time, the trails that remain stronger (reinforced by other ants) are those which are travelled most frequently. These, on average, tend to be the shortest paths between desirable points, while the longer trails require more effort to maintain (tending to weaken). Hence, the general problem-solving approach can be described by just a few simple behaviors (random exploration and conditional trail setting/following).



Science/AAAS, http://www.wired.com/2010/01/slime-mold-grows-network-just-like-tokyo-rail-system/

Slime mould is an example of another organism that has been used to explore shortest-path algorithms. Fig () shows slime mould growing to reveal efficient transport routes between significant locations around Tokyo.

Honey bees themselves have come under significant stu.. the travelling salesman problem (a common problem for which significant financial interest exists for finding an effective algorithm).[[18]](#footnote-19) A group at Rothamsted Research recently conducted an experiment to determine what rules bees are using to narrow down solutions effectively as they move from flower to flower. The results were published in 2012.[[19]](#footnote-20) While it was found that bees do not always find the most efficient solution, they do employ an effective heuristic approach to home in on a good enough solution, quickly.

Unsurprisingly the development and optimization of computer algorithms is a significant enterprise, with ramification that reach into all areas of society. While the design and structure of modern computer algorithms is within the realm of human comprehension and ability, the speed and effectiveness with which such algorithms are capable of solving hugely complex problems certainly is not.[[20]](#footnote-21) This is a field that is anticipated to have an increasingly dramatic societal influence, and we therefore deem this to be a highly relevant and appropriate topic.



https://en.wikipedia.org/wiki/Swarm\_behaviour#/media/File:Auklet\_flock\_Shumagins\_1986.jpg

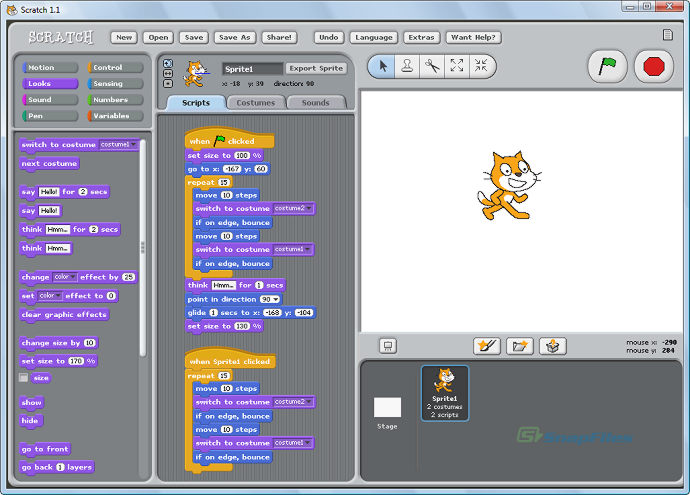
Equally important to the success of our game, and its ability to inspire, is that swarm structures are often enthralling to observe; the flocking behavior of birds (see fig()), shoaling fish, or foraging ants. Our hopes are that swarm structures will be equally entertaining to manipulate and design.

### Currently on the Market 2.3.2

In order to effectively focus the player’s attention on the logic and structure of their algorithm, we have decided to move away from text based programming systems. For absolute beginners these low error tolerance systems are unlikely to provide the level of “forgiveness” recommended by our ICT contact at Bristol Grammar School. In relation to this point, the head of ICT introduced us to Scratch (MIT’s visual programming language) and explained that its user-friendly, graphical programming approach is popular with pupils at the school.

This section will analyse relevant games and technologies, such as Scratch, in order to identify areas that we should focus on in the development of our game should to achieve a level of differentiation.

### Scratch 2.3.2a



"Scratch is developed by the Lifelong Kindergarten Group at the MIT Media Lab. See http://scratch.mit.edu."

Scratch (first launched in 2003)[[21]](#footnote-22) allows users to easily manipulate sprites using a drag and drop, block based programming language. Sprites are first selected using the mouse and instructions for that sprite are then issued by dragging and arranging code blocks in the editing pane. The language is very popular in educational contexts because of its easy-to-use, intuitive interface. Additionally Scratch’s remix feature allows users to build on and learning from the projects that have already been created by other users.

The scratch editor is sufficiently flexible to allow users to create sprites with simple AIs that are able to respond to changing contexts and actions (requiring a large number of if-else statements). Such scripts are typically very long and tend to be complicated for users to understand. As mentioned on the website this “makes remixing a problem. Because of all this, most AI projects have no improvements, causing the AI to remain glitchy.”[[22]](#footnote-23)



<http://snap.berkeley.edu/run>

Snap! is developed by Jens Mönig with the help of Brian Harvey and an ever-expanding team of Scratchers. It is sponsored by the University of California, Berkeley, with support from the National Science Foundation and the MioSoft Corporation.

A particular technique used by Scratch (and other similar software such as Snap, a scratch derivative) to make flow control structures easier to recognize and follow, is to highlight the edges of affected blocks, and nested groups of blocks, with the same colour. These languages are multipurpose, affording the user flexibility to lots of different things. In our case, the intention will be to specialise in facilitating potentially convoluted flow control structures. This will be necessary for the construction of algorithms that are sufficiently flexible to deal with changing game contexts.

### Educational Coding Games 2.3.2b

Similar visual, educational programming languages to Scratch include Snap[[23]](#footnote-24) (a Scratch derivative launched in 2009), Kodu[[24]](#footnote-25) (2009), and Alice[[25]](#footnote-26) (1998). These all share similar features and are designed to allow individuals to create their own software, games or animations (rather than acting as self-contained or well-defined games themselves).

The majority of games based on visual programming languages that we have come across are aimed at 5-9 year olds. Such games tend to be relatively static puzzle games where instructions are relayed, command by command, to manipulate the state of a specific puzzle. These include lightBot,[[26]](#footnote-27) Kodable,[[27]](#footnote-28) Robozzle.[[28]](#footnote-29) CargoBot[[29]](#footnote-30) (2012) is a game that markets itself for all ages but follows the same relatively static logic of game play. ‘Code Warriors: Hakitzu Battles’[[30]](#footnote-31) (2013) is a related game, aimed at ages 9+, but it is based on combat as opposed to puzzle problems. Players issue commands using a set of functions. The commands are text based (requiring exact syntax) although the same sequential, command by command, move by move approach of the previous games is used, with instructions tailored to a relatively static context.

These systems appear to leave little room for programming condition-based autonomy (to deal effectively with changing contexts in the game). This is intended to be a key feature in our game, although it will require a well-designed algorithm editor/visualiser, as mentioned in the previous section.

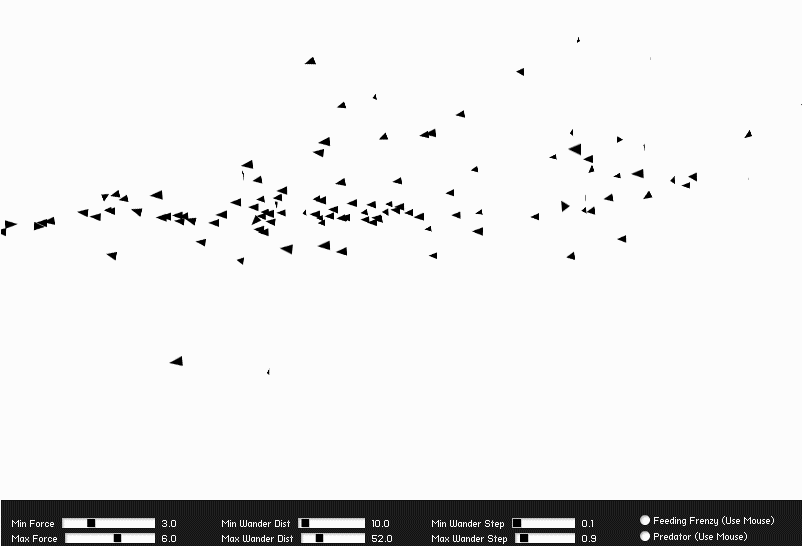
### Swarm Manipulation 2.3.2c

Aside from the educational focus of our game, a primary feature will be the ability to design and manipulate the behaviour of a swarm. There are a few freely available games on the market that allow players to manipulate swarm behaviour, in order to achieve certain goals. Swarm Control (2012, by Armour Games)[[31]](#footnote-32) requires the player to protect a bee-hive by instructing a swarm of bees to attack approaching bears. This is instruction is issued by the player conventionally, using a click and swipe actions with the mouse. The game therefore differs to our concept in terms of the mode of player-swarm interaction, and consequently, the complexity of instructions that can be issued.

Another similarly named game, SwarmControl (a 2013 research project by Aron Becker and Chris Ertel)[[32]](#footnote-33) builds on this functionality, giving the player an option to globally or locally attract or repel particles in order to solve a range of different challenges and puzzles (such as rounding up other objects and moving through mazes). Similarly to the previous game, instructions are issued conventionally (using the mouse) and the complexity of instructions are limited.

Other approaches to manipulating swarms tend to present in the form of simulators as opposed to well-defined games. These allow a greater variety of instructions that can be issued to a swarm, using sliding bars or value inputs to alter variables that describe the behavioural rules.

Boid[[33]](#footnote-34) (2009) is an example of one of these systems. While the user is able to alter the actual algorithm that controls the swarm, this manipulation is limited to numeric variables rather than the insertion of new rules or the definition of Boolean conditions. While more sophisticated simulators than Boid will certainly exit, in our search we have not come across any that provide the level of swarm control, in the form of a game, that we intend on providing our players.



Boid, http://blog.soulwire.co.uk/laboratory/flash/as3-flocking-steering-behaviors

There appears to be a gap in the market for a game that provides an educational coding experience, while providing appropriate control over swarm behavior.

## Final Concept 2.4

The player’s objective in Code A Colony is to design and build a sufficiently complex algorithm to allow their colony to survive for as many years as possible. This will require the bees to explore, to discover and collect resources (enough to brave the winter), and to communicate information to other bees, while also fending off enemies and responding to environmental threats such as rain and snow.

The ubiquity and real-world impact of problem solving algorithms, along with their potential definition by relatively simple rules, makes this an appealing educational topic. Our intention is to develop a system to educate and inspire school children about the underlying mechanism of problem-solving algorithms, and to inspire the next generation of computer scientists and engineers to design and discover the world shaping algorithms of the future.

### Unique Selling Points 2.4.1

In response to our market research, we have formulated a series of features that we intend on emphasizing as unique selling points of our concept. As a means of differentiating ourselves from other educational coding games, our system will:

* Facilitate the construction of complex conditional algorithms (built from simple elements) that are capable of effectively driving autonomous actions in different contexts. The player will spend their time programming decision-making structures rather than defining a static sequence of behaviours for very specific context.
* Provide a block based programming interface that facilitates intuitive comprehension of nested/convoluted flow control structures, so that flexible and dynamic algorithms can be created to drive autonomous behaviour. We will taking some inspiration from Scratch’s drag and drop system, though this is acknowledged to have limited scope for programming autonomous behaviour in an easy-to-read, digestible manner.
* Provide the player with significant control over a colony of bees, allowing them to experiment flexibly with different swarm behaviours and to observe the results of their design at both a local and global scale (close-up view and map view).

**Requirement Specification 3.0**

## Introduction 3.1

The Software Requirement Specification (SRS) which follows, is designed to detail how our project is to be implemented, with reference to the scope and purpose of the software, and how that lines up with the requirements of first, the team, and second, the client – and concordantly the target Audience they seek the serve.

## Overview 3.2

Our vision for the game is a system by which a user can control the movement of a swarm of bees using block-based programming. In line with our market research [section 1.4.1], the world is intended to be explorable, non-static, interactive, and open. By the same token, we aim to deliver attractive graphics of a cartoon nature, and an immersive and progressing story with measures of success enabling players to compete with one another. And as advised by the teacher we spoke with, the game should in its nature be forgiving, responsive, and deep.

## Product Components 3.3

The system proposed to address the requirements will have features familiar to users as those of a game: an opening title screen; ongoing sound tracks; informative sound effects; a heads-up display (HUD); a navigable world-map with terrain features; moving characters in the guises of protagonist, adversary, and benefactor; timed events; and interactive objects.

### Opening title screen 3.3.1

As the first view of our product, it is necessary for the title screen to entice users to play the game by showcasing its story and gameplay in a lively manner. It should also accommodate buttons of different utility: at a minimum, a button to start the game, and another to explain it.

### Audio 3.3.2

While audio was not ranked exceptionally high as a valued factor in our market research, we feel it is integral to setting an immersive story (through soundtracks), and providing feedback for actions (through sound effects), which is a key factor in game responsiveness. Audio can also provide an element of humour to make the game more engaging.

### HUD 3.3.3

The HUD is crucial for co-ordinating the movement and status of the hive and is thus the main facilitator of interaction with the game. We specify, as a minimum, the necessity to be able to: keep track of resources; make purchases with those resources; display announcements of timed events; keep track of time remaining until story events; mute audio; view the whole world map in miniature; and above all, program the hive.

### World and terrain 3.3.4

The world should be: large and unbounded enough to be explored freely (an ‘open world’); procedurally generated to increase replayability (giving the game depth); reflective of the story (ie. including grass, flowers, and trees) to aid immersion; and visually attractive (to satisfy the value placed on graphics). The popularity of non-static terrain as a feature also inspires a need for resources such as flowers to deplenish and replenish, and non-interactive scenery like trees and grass to change in appearance over time.

### Characters 3.3.5

Characters are integral to telling stories and immersing through their capacity for humour. By preparing a protagonist (the bee), an adversary (a spider), and a benefactor (“Garry the Ice-cream Man”), we can also achieve a greater level of depth through their interactions.

### Timed Events 3.3.6

We can increase the depth of the game by introducing characters into play through timed events (the adversary and benefactor). Periodic weather effects impacting on the protagonists could serve this purpose. Visible changes in season, culminating in a winter by which time the protagonists must have sought shelter, would also provide a solid framework for a story.

### Interactive objects 3.3.7

The responsiveness of the game depends on interactivity, which, beyond the ability to program how the protagonist interacts with objects, can be provided through an ability to track moving objects by clicking on them.

## Use Characteristics 3.4

The fundamental characteristics of our central target audience are as follows:

* They are Key Stage 3 Learners (between ages 11 and 14).
* They have varying degrees of previous experience in programming.
* They are competent in more general purpose usage of computers.
* They are of mixed genders.

Post-Note

Whilst these characteristics represent the core target audience, as the product enters its alpha stage of development (with a working framework for all systems, and more-or-less feature completeness) we feel the appeal of Code A Colony may have become somewhat broader, encompassing slightly older and indeed even younger students. This is a facet of its depth combined with its simplicity; more practiced, older students may still take something away from playing the game, while those younger and completely new to coding will quickly be able to pick up the basics of the game, and progress from there.

## Dependencies 3.5

The code will be developed from day one on several different platforms owned by the coding team: Windows 32-bit and 64-bit; Ubuntu 64-bit (as compatible with the university’s lab workstations); and Mac OS X 64-bit. Builds for any of these platforms can be produced by running their respective makefiles (or cmakelists file, in the case of Mac OS development). Internet access is not required in-game. We will make use of graphics acceleration, so OpenGL may need to be installed by the user.

Trackpad support will vary based on trackpad driver (right-click-dragging is supported on the Microsoft Surface Pro 4, but not on an Apple Macbook Pro), and stylus/touch support will vary in the same respect based on driver and right-click behaviour setup. Development for mobile systems would be possible (and attractive, given the ubiquity of mobile devices among our target market), although we lack access to suitable development hardware on which to collaborate, so provisionally can’t support them.

## Constraints 3.6

### Time 3.6.1

From the outset, the team is very conscious that our main constraint will inevitably be time. Whilst initial research and conceptual design started in Early October, the development cycle will only really begin in earnest come November, leaving us approximately three months to go from writing the first line of code, to exhibition of a final demo.

### Skillset 3.6.2

It is clear that another constraint, tied inextricably to the first, will be that the team is still only now beginning to get to grips with the central technologies the game is to utilize, namely the programming language C, in concert with the library SDL and other subcomponents thereof. We expect this to make the already problematic time constraint all the more challenging to work within; early development will likely be hampered and stop-start as we collectively struggled to familiarize ourselves with the tools for execution of our ambitious designs. Fortunately, we have within our team a wide range of abilities in graphic, video, and audio design going in, so work in those areas should be more manageable.

### Fair Use and Copyright 3.6.3

Whilst we expect at least initially to use placeholder art and sound from “the internet”, we will ultimately need to create our own assets for the project in order to remain within legal constraints. Again this will likely be a time-consuming element of production.

### Divided Focus 3.6.4

While we hope to be able to dedicate ourselves to the product’s development, we appreciate that we there will be periods where the project will be only a secondary priority, as we switch our focus to other parts of the gruelling Computer Science Programme at the University of Bristol at various points between now and the project deadline. The extent of this is difficult to predict, as we cannot guess at how taxing other units will become as time progresses.

## Quality Assurance 3.7

### Suitability 3.7.1

The prototype must conform to the requirements of our client – teaching or enthusing youths about computer science. First, we will have to remain mindful of our goals, keeping the purpose in the forefront as our project is moulded. Second, we will need to conduct user-testing and seek informative feedback from our target audience to ensure we are on target.

### Efficiency 3.7.2

The prototype must not be excessively demanding. Whilst our project could certainly become a behemoth, dominating system resources due to its very nature – seeking to emulate and control complex swarm behaviour – we will make every effort to keep computation minimal wherever possible. Our metrics for this will be framerates, responsiveness, and simple system monitoring utilities such as *task manager* and *GPU-z[[34]](#footnote-35)*.

### http://www.giga.de/wp-content/gallery/gpu-z_1/gpu-z-1.jpg?PageSpeed=off&PageSpeedFilters=-resize_images,-resize_rendered_image_dimensionsFlexibility 3.7.3

Figure 3.7-1GPU-Z monitoring of GPU sensors.

The prototype needs to be developed in a malleable manner, such that we can adapt the product as we receive feedback and tweak the direction of the product from time to time. In real terms a large part of this will be in the modularity of the title, and making sure dependencies between said modules are manageable. If changing one module a little, entirely breaks the others we’ll have failed to some extent in this regard.

### Security 3.7.4

We do not expect security to become a concern for our program which will be run entirely locally.

### Maintainability 3.7.5

Once we have delivered the product, we do not expect to be maintaining it, but none-the-less, good coding practice is to endeavour to make the code accessible to new eyes, unfamiliar with the project. To this end we will develop and be consistent to a team style-guide. This will include how we name variables, as well as where and when to include comments, and at what level of detail.

### Portability 3.7.6

As we all have different native operating environments in which we will be developing (Linux, Macintosh, and Windows) it will be important from the off that we ensure our code is easily portable between platforms. The metric for judging this will be simply whether or not we can consistently compile and run it on these different OSes.

### Stability 3.7.7

The prototype should be stable. This means it should crash rarely if ever, and certainly any reproducible bugs should be fixed at the earliest opportunity. Testing should as far as possible explore all possibilities for the ways a user might interact with our product, and if we can take the software to its very limits without breaking it, we can be satisfied in its stability.

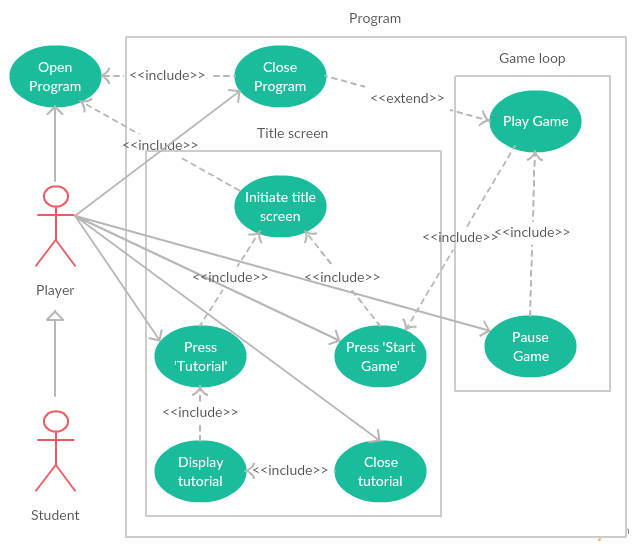
### Usability 3.7.8

The prototype must be accessible to users. It would evidently be pointless to build a flawless swarm emulator, with complex and complete user control of the underlying AI if the user is then unable to engage with a cryptic user interface. Ergo usability must remain at the forefront of the team’s consciousness throughout design and implementation. This will be another area where user-testing a feedback will be crucial to success.

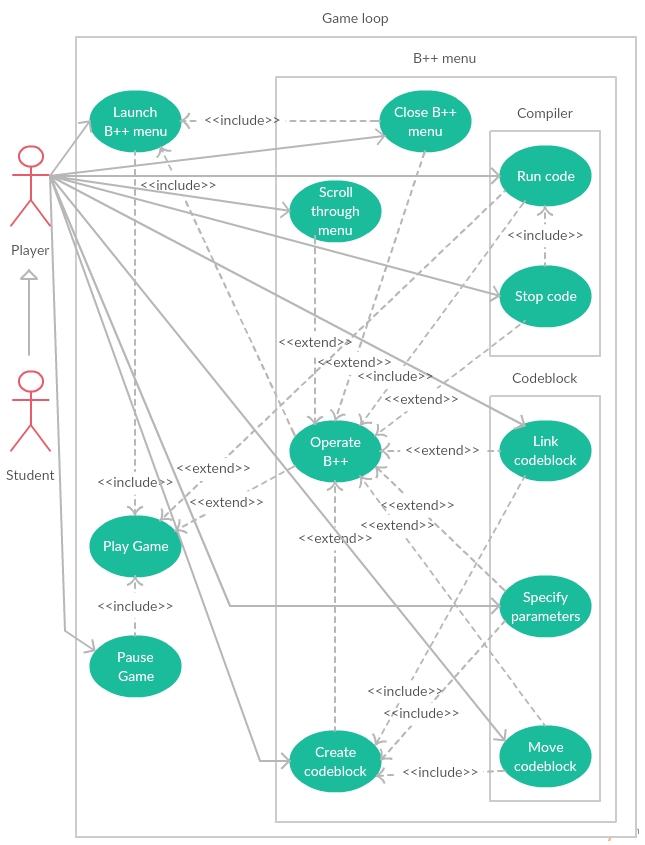
## Use Cases 3.8

The planned usage scenario is:

The player, a student, will open the program to be faced with the title screen, whereupon they can choose with a button press whether to view a tutorial or start the game itself. Viewing the tutorial displays an explanatory image, which can be closed once satisfied. By choosing rather to start the game, the game loop will begin. The game loop can be paused at any point with the input of a pause key. At any point after opening the program, the user may also close it by cllicking the ‘X’ button in the corner of the program’s window.



B++ usage:



**Design Implementation 4.0**

## Module Design 4.1

## Systems Architecture 4.2

## Graphics 4.3

Being a core interest implicated in our market research, great attention was dedicated to the graphics module of the game. Graphics usages included: the splash screen; the game world; the objects within the game; and the user interface. Each of these graphics applications went through a design process involving input from the whole group.

### Opening title screen 4.3.1

Concept sketches were produced and put up for team input. One design was favoured and further developed into a colour sketch, which was then translated into a digital painting; details like scenery were developed upon with input from group members.

[include scans of sketches]

### Game world 4.3.2

The game world consisted of a large grass background stretching beyond the extents of the viewing window which could necessarily be scrolled along. Initially, a green rectangle was used, but this was later replaced with a repeating drawn grass texture. This original grass texture unfortunately didn’t tile well, so the final grass used in the game is a placeholder: a stock image sourced from Google Images.

### Game objects 4.3.3

Placeholder stock images sourced from Google Images were again exploited for flowers, the spider, and ‘Garry, the ice cream man’. The hive image was drawn without considerable development, while the stars of the game – the bees – were developed through a series of sketches put up to a vote. The accessory flight goggles indicating that the bee was part of the initially planned ‘scout class’ remain in use for all bees, as a class system was ultimately never implemented in the final product. The decision to leave the goggles on display was simply that we felt their mere presence added character to the bee.

### User interface 4.3.4

Design of the user interface was delegated to one member in the group, then implemented by another. This initial implementation suffered from an illegible font, lack of colour hierarchy and co-ordination, and unlabelled buttons. Incremental changes to the interface (based on in-practice navigability and user-friendliness) were gradually made with input from everyone, resulting in a compact and responsive interface that was intuitive enough for the target users to operate when finally exposed to the product [see section \_\_\_ ].

## Audio Design 4.5

For audio creation we utilized two approaches. For most sound we used tools to create all new sounds for the project. The exception is In the case of one particular sound which we cut from the popular TV series *“Game of Thrones”* as a deliberate comedic Easter Egg[[35]](#footnote-36).

The first audio we set out to implement was the music – a decision we latterly regretted as it began to haunt our dreams. Initially we began this process using the open source tool Audacity (figure 4.5-1), but due to the stylistic choices we were making – with a retro 16-bit, Amiga-like feel, to complement the cartoon graphical style – it became clear the software wouldn’t suffice. Instead we moved to Cakewalk’s Sonar X3 Audacity (figure 4.5-2), a professional suite with stronger MIDI support, and more nuanced and powerful post production capabilities. The tracks that play after the splash screen is dismissed are all designed as an homage to early 90s games, which would often cover, and slightly jazz up, popular hits[[36]](#footnote-37). Though they can repeat with enormous frequency, even during just a single play-through given their small number, they are catchy, and enjoyable, and the melodies sneakily keep the game in your consciousness even hours later. The three songs shipping with the game are reimagined versions of “*Ain’t No Sunshine[[37]](#footnote-38)*”, “*Summertime[[38]](#footnote-39)*”, & *“Black Horse & A Cherry Tree[[39]](#footnote-40)”.*

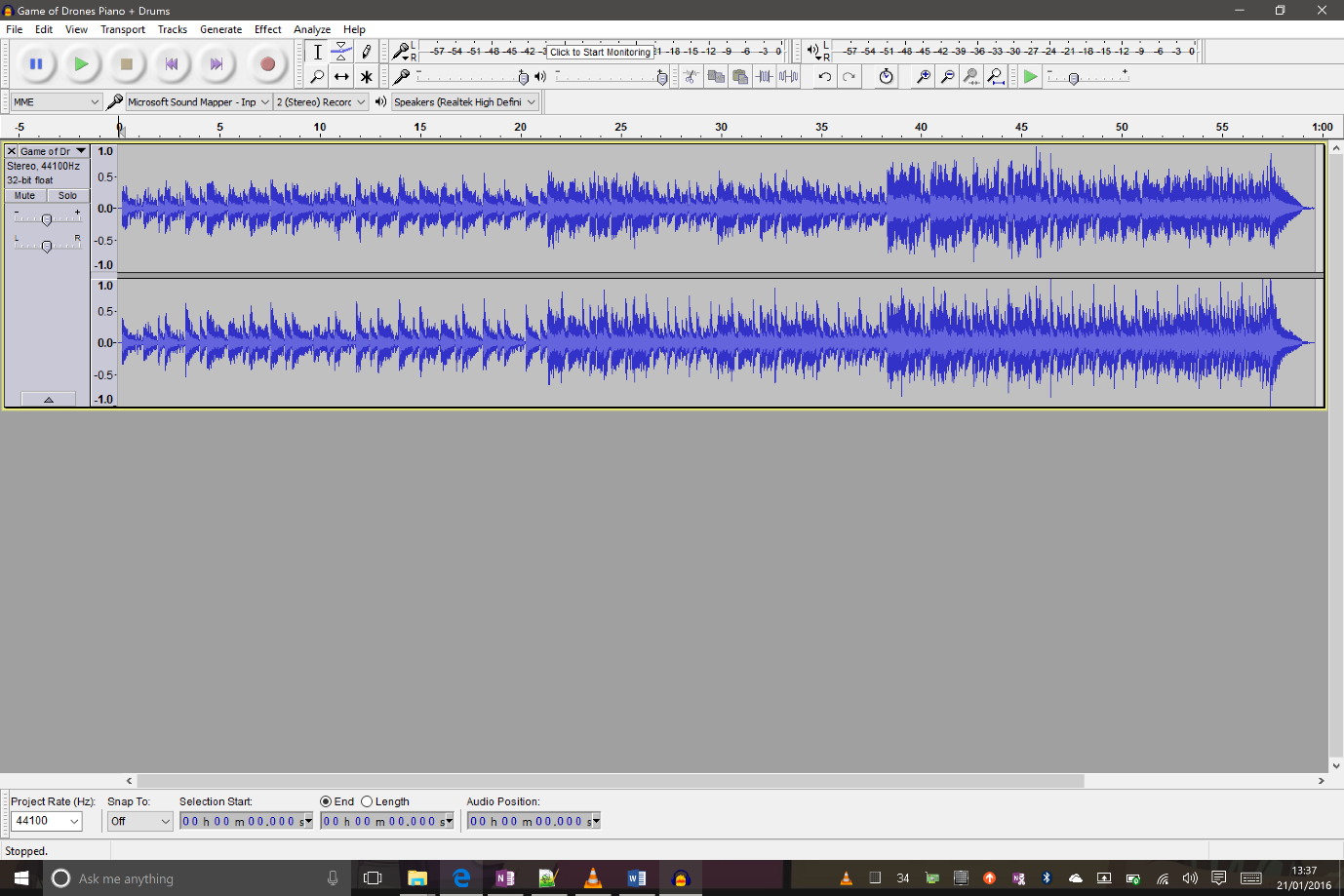


Figure 4.5-1: Audacity

The theme song which plays upon initial game launch is another homage, in equal parts, to *“the Frog Song[[40]](#footnote-41)”*, and again to Game of Thrones, owing to an early jocular working-title of the game: *“Game of Drones”*. Whilst the other tracks required principally the use of a midi-keyboard and drum-kit, the idea of the theme was to mimic the hum of a swarm. For this we simply layered ourselves ‘buzzing’ the theme, and layered various recordings on top of each other to give the sound a depth to simulate the swarm.

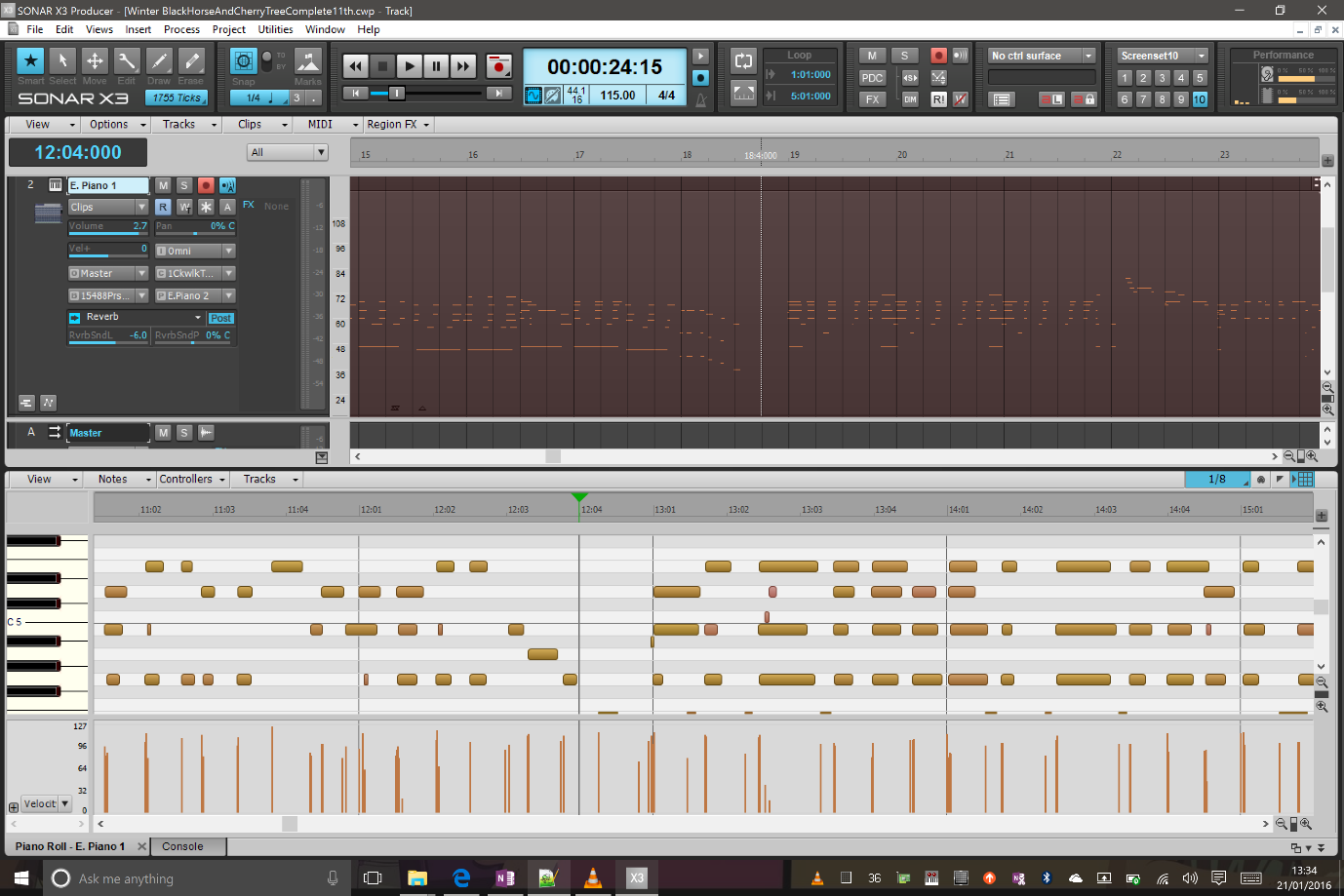


Figure 4.5-2: Cakewalk Sonar X3

The remaining sound effects integrated include a midi-piano-generated chirp, played when the user increments his honey counter (score) by 50, a rain sound effect for when it is raining, taken from SoundBible (a royalty free sound bites project at [*http://soundbible.com/royalty-free-sounds-1.html*](http://soundbible.com/royalty-free-sounds-1.html) ) and a scream for when the Ice-Cream-carrying-victim is stung. For this last sound effect, we asked (with the teacher’s permission) if we could get member of the class who were part of our focus group to record a scream, with the promise that we would integrate the best one into the game.

## Core Features 4.6

## Testing 4.7

We have adopted the unit testing framework check to perform testing. Check is from the webpage (http://libcheck.github.io/check/).

The scope of testing included (i) a full scope testing for all functions in generic.c and control.c, and (ii) limited testing on selected functions in other .c files.

To test a function which returns an output (could be a predefined type or a struct), we would check if the output of the function match our expected value or not. If it matches our expected value, it passes the test. Otherwise, the test is failed and we need to investigate the reason of failing the test.

Some functions are void functions thus not returning any value. In order to test these functions, we would investigate the changes carried out by these functions. The testing method is to check the expected value of certain variables after performing these functions.

Some functions involved inputs and/or outputs SDL type of variables. Thus, we need to initialize these variables properly before testing these functions. For function output to the screen directly, we would need to test them using methods similar to testing void functions.

## Tools 4.8

### Collaboration 4.8.1

For coding, we used GitHub (see section 4.8.4) as a version repository through which we could generate branches from a master version of the code to then be edited and later pushed (and merged when necessary) as a newer version. Despite its steep learning curve, we all came to value the power it offered in enabling multiple people to work with minimal impact on each other and its capacity to rewind any ill-made edits.

For creating presentations, we used Google’s Cloud-based office suite, allowing simultaneous, instantly pushed and pulled edits. While limited in capabilities, it proved a very streamlined tool for collaboration.

For this report writeup, we hosted the report on Microsoft Onedrive, allowing simultaneous, albeit evidently not instant, edits (which caused some grief and thus removed it as a candidate for future collaborative work) using Microsoft Word.

### C 4.8.2

Coded entirely in C, the tools we chose a variety of tools to ease working with the language.

* **SDL2 and associated libraries:** SDL was the graphics library of choice in our Programming in C module, so we were happy when it was specified as the suggested graphics library to be used in this project’s module. It proved to be fairly easy to use once accustomed to, and by compiling the SDL2, SDL\_ttf and SDL\_mixer libraries for the lab machines, we were able to expand our usable featureset to include text manipulation and audio. However, constantly linking new libraries for each of our development platforms proved burdensome, so incorporation of additional libaries such as SDL\_image (for advanced image features like alpha channels and PNG support) had to be abandoned to keep the project moving forward. Licenses: zlib.
* **Unit testing framework:** We chose Check for its very intuitive appearance, and for it being pre-installed on the lab machines. It proved to have a fair featureset (able to divide test-cases into sets of tests and suites, and has ‘forking’ to enable continuing tests despite segfaults, unlike the rival framework CUnit), although the pitifully low amount of documentation meant that we lacked any knowledge of certain desired features such as how to set up mocking for tests. License: LGPL.
* **Debuggers:** Simply those that were ready integrated into our development environments: on Linux and Windows, GDB; and on Mac, LLDB. Licenses: GNU GPL and LLVM, respectively.
* **Memory management:** Where memory leaks and segfaults were of concern, Valgrind was used for being well-known and pre-installed on the lab machines. License: GNU GPL.

### Media 4.8.3

Our team used the tools we were most familiar with: for digital art, Celsys Clip Studio Paint; for image editing, Adobe Photoshop and GIMP 2; for video creation, Microsoft Movie Maker; and for audio, Cakewalk Sonar X3 (and briefly Audacity).

### Version control 4.8.4

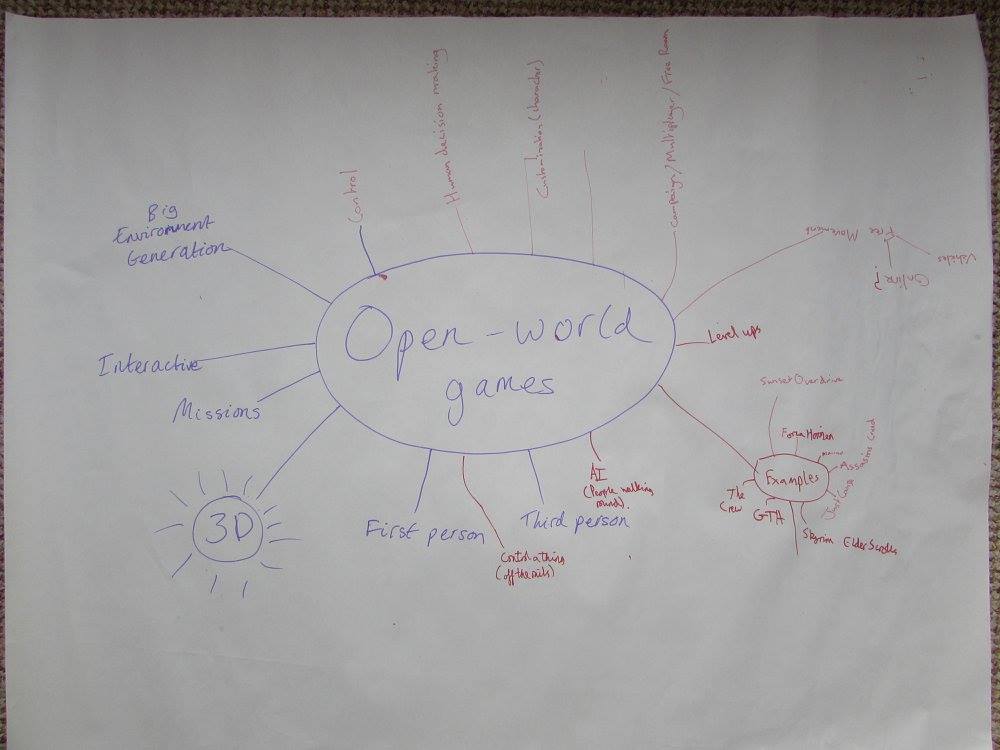
We used ‘git’ as a version control system, pushing remotely to GitHub due to having an existing university account for it that could host a private multi-user repository, unlike the public ones we’d generally be limited to with other services, and also due to the advantage of being able to accustomise ourselves with such an industry standard. While the learning curve was steep, it worked well as a version control tool, and we expect the experience would have been largely the same no matter what remote service had been used. Since we were more familiar with git than alternatives such as SVN, we were content with our choice of underlying system too.

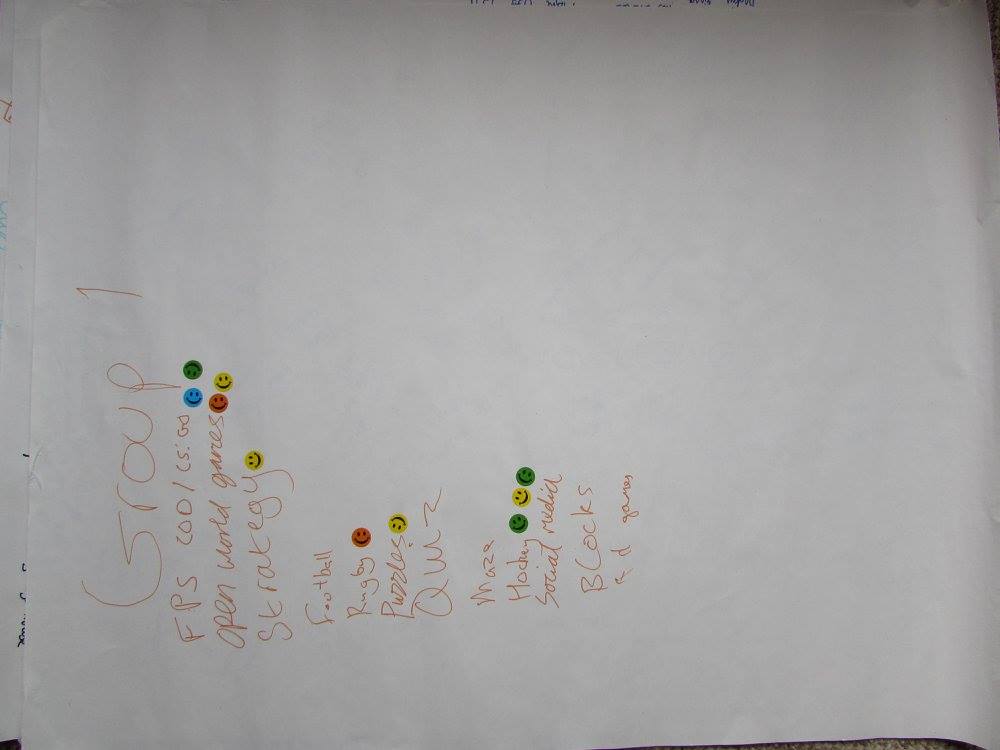
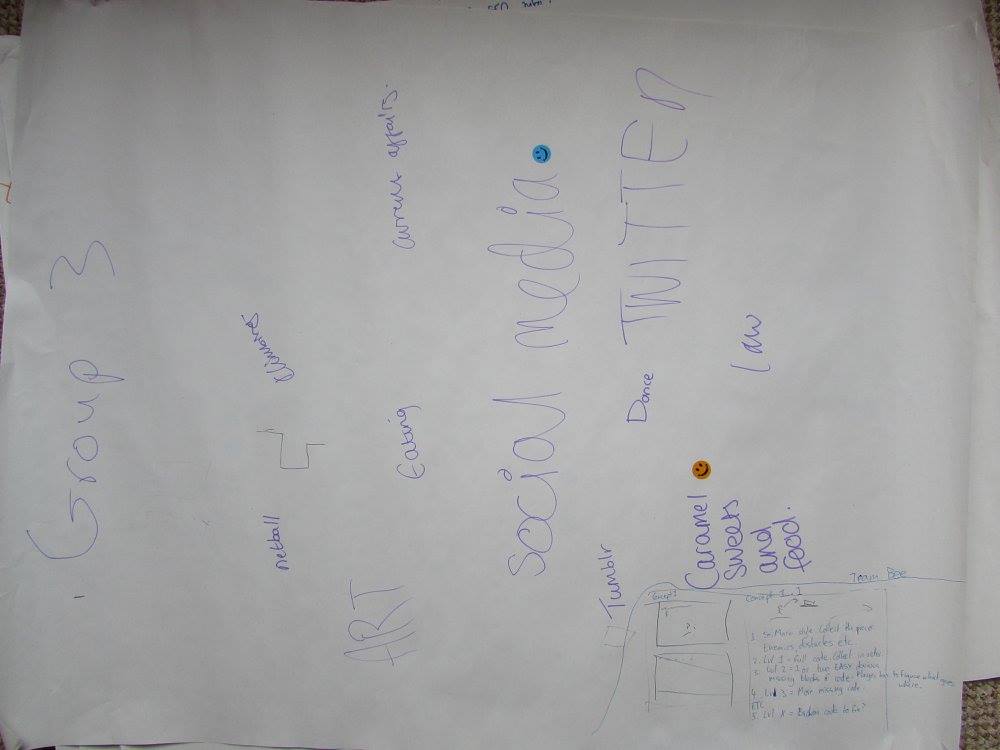
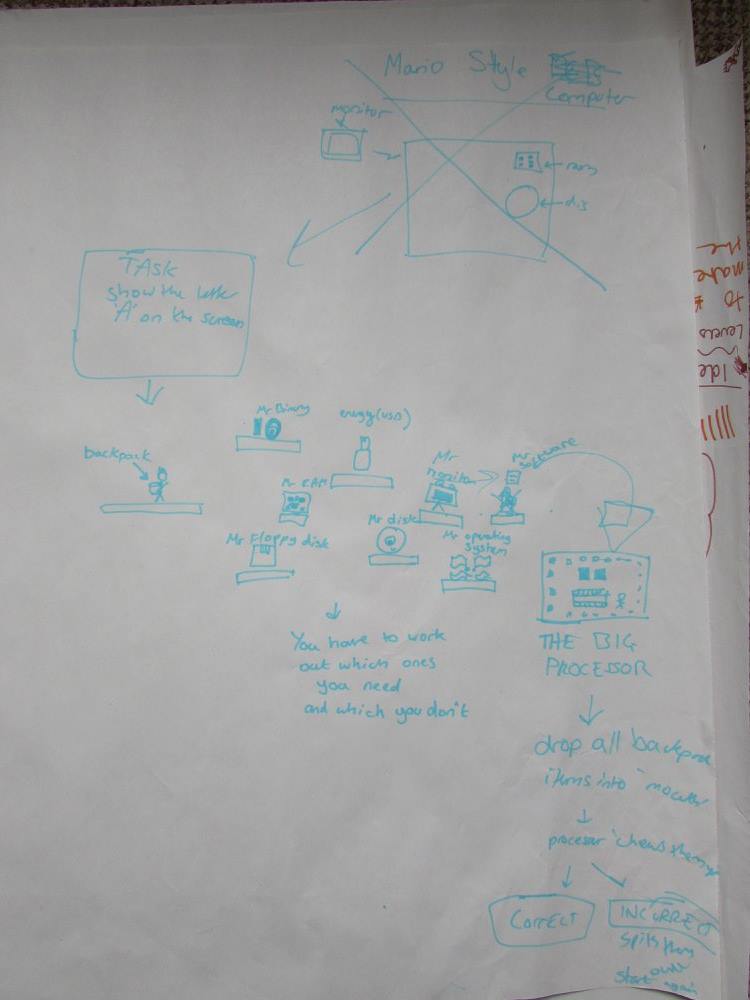
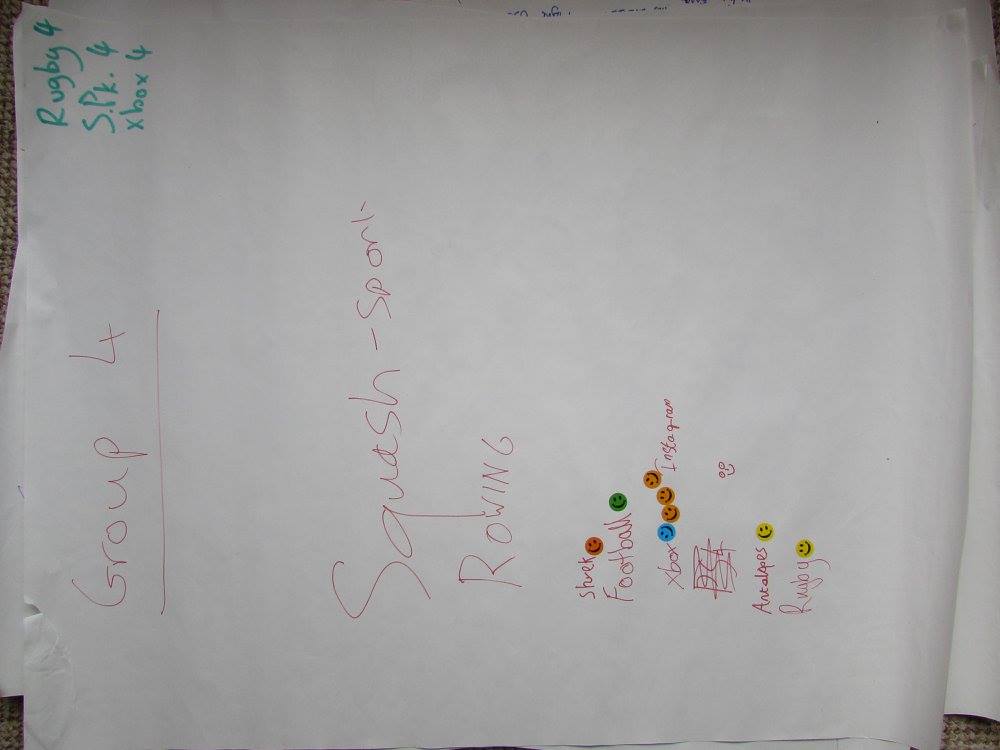
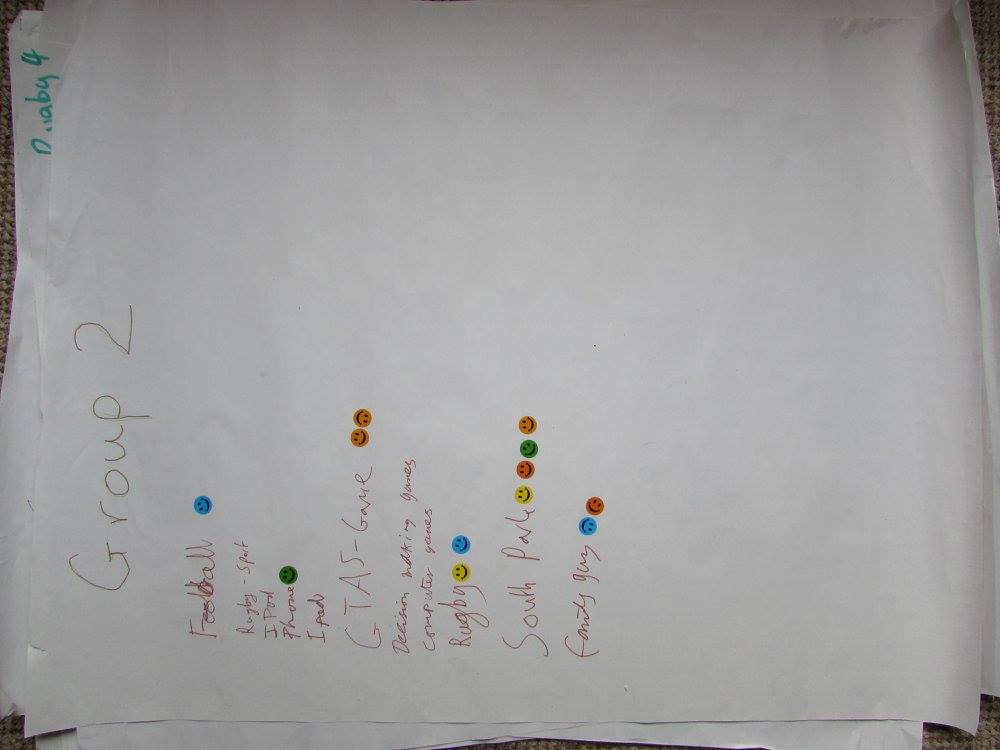
**Planning 2.**

**Project Management 3.**

**Objectives have been moved into introduction section, altered to be less generic**

Appendix 1

****

****

1. https://www.nesta.org.uk/sites/default/files/the\_legacy\_of\_bbc\_micro.pdf [↑](#footnote-ref-2)
2. http://www.ebuyer.com/blog/2015/07/can-the-bbcs-micro-bit-inspire-a-generation/ [↑](#footnote-ref-3)
3. http://www.theguardian.com/media/interactive/2011/aug/26/eric-schmidt-mactaggart-lecture-full-text [↑](#footnote-ref-4)
4. https://www.gov.uk/government/uploads/system/uploads/attachment\_data/file/239067/SECONDARY\_national\_curriculum\_-\_Computing.pdf [↑](#footnote-ref-5)
5. https://www.gov.uk/government/uploads/system/uploads/attachment\_data/file/239067/SECONDARY\_national\_curriculum\_-\_Computing.pdf [↑](#footnote-ref-6)
6. http://www.computerweekly.com/feature/The-UKs-new-computing-curriculum-is-here-Are-teachers-ready [↑](#footnote-ref-7)
7. http://schoolsweek.co.uk/teachers-lack-confidence-in-computing-science/ [↑](#footnote-ref-8)
8. http://schoolsweek.co.uk/3m-on-and-where-are-all-the-master-teachers/ [↑](#footnote-ref-9)
9. http://www.theguardian.com/technology/2014/sep/04/coding-school-computing-children-programming [↑](#footnote-ref-10)
10. https://petition.parliament.uk/petitions/111693 [↑](#footnote-ref-11)
11. http://www.bbc.co.uk/news/technology-29010511 [↑](#footnote-ref-12)
12. <http://www.nesta.org.uk/sites/default/files/next_gen_wv.pdf> p.38 [↑](#footnote-ref-13)
13. <http://www.nesta.org.uk/sites/default/files/next_gen_wv.pdf> p.5 [↑](#footnote-ref-14)
14. <http://www.nesta.org.uk/sites/default/files/next_gen_wv.pdf> p.38 [↑](#footnote-ref-15)
15. https://www.gov.uk/government/uploads/system/uploads/attachment\_data/file/239067/SECONDARY\_national\_curriculum\_-\_Computing.pdf [↑](#footnote-ref-16)
16. http://www.bbc.co.uk/iplayer/episode/b03k6ypz/the-joy-of-logic [↑](#footnote-ref-17)
17. http://www.bbc.co.uk/programmes/p030s6b3 [↑](#footnote-ref-18)
18. Secrete rules of modern living, bbc, marcus du sautoy [↑](#footnote-ref-19)
19. http://journals.plos.org/plosbiology/article?id=10.1371/journal.pbio.1001392 [↑](#footnote-ref-20)
20. Secrete rules of modern living, bbc, marcus du sautoy 00:48:35 [↑](#footnote-ref-21)
21. http://wiki.scratch.mit.edu/wiki/Development\_of\_Scratch\_1.0 [↑](#footnote-ref-22)
22. http://wiki.scratch.mit.edu/wiki/Artificial\_Intelligence [↑](#footnote-ref-23)
23. http://snap.berkeley.edu/run [↑](#footnote-ref-24)
24. http://blogs.msdn.com/b/ukschools/archive/2010/04/15/kodu-for-pc-a-teacher-s-tutorial.aspx [↑](#footnote-ref-25)
25. http://www.alice.org/index.php?page=what\_is\_alice/what\_is\_alice [↑](#footnote-ref-26)
26. https://lightbot.com/hocflash.html [↑](#footnote-ref-27)
27. https://www.kodable.com/ [↑](#footnote-ref-28)
28. http://www.robozzle.com/ [↑](#footnote-ref-29)
29. http://twolivesleft.com/CargoBot/?utm\_campaign=Listly&utm\_medium=list&utm\_source=listly [↑](#footnote-ref-30)
30. http://www.codewarriorsgame.com/kai-web-kai/login [↑](#footnote-ref-31)
31. http://armorgames.com/play/12841/swarm-control [↑](#footnote-ref-32)
32. http://www.swarmcontrol.net/ [↑](#footnote-ref-33)
33. http://blog.soulwire.co.uk/laboratory/flash/as3-flocking-steering-behaviors [↑](#footnote-ref-34)
34. GPU-Z is a tool for monitoring graphics card variables, such as temperature, clock-speed and load. [↑](#footnote-ref-35)
35. Game of Thrones, Season 1, Episode 1: Winter is Coming – Quote: “Winter is Coming.” [↑](#footnote-ref-36)
36. For example, Amiga game Rainbow Island’s 16-bit version “somewhere over the rainbow”. [↑](#footnote-ref-37)
37. Ain’t No Sunshine (1971) – Original Artist: Bill Withers [↑](#footnote-ref-38)
38. Summertime (1934) – Original Artist: George Gershwin [↑](#footnote-ref-39)
39. Black Horse & A Cherry Tree (2005) – Original Artist: KT Tunstall [↑](#footnote-ref-40)
40. Paul McCartney – Rupert and The Frog Song (1986) – Animated Movie [↑](#footnote-ref-41)