# Software Design and Development

# Higher School Certificate

Major Project
Stages 1, 2, and 3

Note: Official submission is Stage 3 and will include all previous stages.

**Teacher - Jay Fairweather**

Rev: 06/10/22

# Contents

## Purpose of the Major Project

The Major Project is designed to assess a student's knowledge, skills and understanding of all aspects of the HSC course. It is intended that the Major Project be a substantive piece of work undertaken throughout the first three terms of the HSC Course. As well as assessing what students have learnt, the Major Project should assist students in clarifying their understanding of key concepts and aid in developing quality responses for questions in the HSC examination.

## Assessment task Value

The Major Project will be allocated ***100*** marks and will constitute ***40%*** of a student's school-based assessment.

## Assessment submission

The Major Project will be assessed in three (3) distinct stages. All stages are **pre-requisite** to stage 3 i.e. it must be completed before submitting the next stage.

The Major Project is cumulative, and students are to submit the progression of their work at all stages (i.e. when submitting Stage 2, Stage 1 is re-submitted).

Stage 1 and 2 can be considered DRAFTS and will not have marks allocated until they are marked at STAGE 3. Instead, at each hand in of stage 1 and 2, a band will be awarded to show your progress.

You may update and improve your stage 1 and 2 submissions.

## Submission Dates

**Stage 1: Understanding the problem, Planning and Designing (50 marks)**
*Uploaded to Newee.io*
*Marking will be in the form of BANDS (1-6)*

**Stage 2: Implementation (25 marks)**
*Uploaded to Newee.io*
*Marking will be in the form of BANDS (1-6)*

**Stage 3: Test, Evaluate and Maintain (25 marks)**
Due no later than 8:30 a.m. on *Monday Term 3 Week 4.*
*Uploaded to Newee.io*
*Marking will be based on the marking criteria with exact marks provided.*

## Outcomes assessed

H1.2 differentiates between various methods used to construct software solutions
H3.1 identifies and evaluates legal, social and ethical issues in a number of contexts
H3.2 constructs software solutions that address legal, social and ethical issues
H4.1 identifies needs to which software solutions are appropriate
H4.2 applies appropriate development methods to solve software problems
H4.3 applies a modular approach to implement well-structured software solutions and evaluates their effectiveness
H5.1 applies project management techniques to maximise the productivity of the software development
H5.2 creates and justifies the need for the various types of documentation required for a software solution
H5.3 selects and applies appropriate software to facilitate the design and development of software solutions
H6.1 assesses the skills required in the software development cycle
H6.2 communicates the processes involved in a software solution to an inexperienced user
H6.3 uses and describes a collaborative approach during the software development cycle
H6.4 develops and evaluates effective user interfaces, in consultation with appropriate people

## Stage 1– Understanding the Problem – Plan the solution – Design the solution

(Software and Course Specifications provided by NESA is the correct guideline that will be used to mark against)

| | | |
|---|---|---|
| Project ideas and decision | What is the problem you are designing the system for?<br>List the ideas, compare pros and cons, and explain your chosen solution. | nil |
| • Interface design – Scrapbook – Ideas- storyboard | The Scrapbook for interface design at this stage should include various ideas and concept sketches. Inclusion of examples of other software interfaces may be used to demonstrate alternatives. These sketches can be hand sketched or via electronic means, but hand sketches must be scanned. | 7 |
| • Research | What currently exist and how has it been done before. Provide examples in the form of screenshots and code snippets. | 6 |
| • Selection of software environment – Reasoning | Why was the software environment chosen? Benefits including demonstration of further research. | 2 |
| • Identification of hardware/software required | The hardware and software required to produce this solution. | 2 |
| • Preliminary Algorithms | Preliminary Algorithms are basic ideas and concept starters outlining your understanding of what is required | 4 |
| • IPO charts | Construct a IPO Chart | 4 |
| • Decision tables/trees | Construct a Decision Table/Tree | 4 |
| • System flowcharts | Construct a System flowchart | 4 |
| • Structure charts | Construct a structure chart | 4 |
| • Dataflow Diagrams | Construct a Dataflow Diagram | 4 |
| • Selection of appropriate data structures (array, record) / Data Dictionary | Data structures are defined with an explanation as to WHY? These were chosen. This will include a data dictionary following standard NESA formatting. | 4 |
| • Gantt Chart | Construct a Gantt Chart based on the planning, design and implementation of your chosen software solution. | 3 |
| • Developer Diary – including communication with clients (teacher) and progress of task / program. | The Developer Diary is the place where all progress, problems and milestones are recorded. This includes notes from meetings. Prepare notes on your current progress. A meeting with your clients (teacher) will need to be made (outside class time) to discuss various issues. | 2 |
| Total Stage 1 marks | | 50 |

## Stage 2 – Implementation

(Software and Course Specifications provided by NESA is the correct guideline that will be used to mark against)

| | | |
|---|---|---|
| • Translation of solution into code | Algorithm to code. The implementation of the code. Includes reporting on how this changed from paper to computer. Internal / intrinsic documentation (commenting) | 9 |
| • Test data | Explain how to choose test data. What data you will use to test this solution and why? | 3 |
| • Online help | Online (any method online) help explaining how to use various functions. | 3 |
| • User Documentation | The **Manual.** Will include installation, help, credits, troubleshooting etc. | 3 |
| • Developer Diary – including communication with clients (teacher) and progress of task / program. | The Developer Diary is the place where all progress, problems and milestones are recorded. This includes notes from meetings. Prepare notes on your current progress. A meeting with your clients (teacher) will need to be made (outside class time) to discuss various issues. | 2 |
| • Video Pitch | Using video – Pitch your new software to a panel of investors. | 5 |
| Total Stage 2 marks | | 25 |

## Stage 3 – Test, Evaluate and Maintain – Final submission.

| | | |
|---|---|---|
| • Program testing | Level of testing, the use of live test data, quality assurance, bench marking. Tabularised information regarding various tests. Desk checking. Reports from various testers including any proformas. | 4 |
| • Social and Ethical issues – How I addressed these issues. | Outline the possible social and ethical issues that could/would influence your solution. | 2 |
| • Maintenance | What maintenance is required, eg, changing, updating? How will it be maintained. | 2 |
| • Prototype | The finished software solution on a USB or UPLOADED | 15 |
| • Developer Diary – including communication with clients (teacher) and progress of task / program. | The Developer Diary is the place where all progress, problems and milestones are recorded. This includes notes from meetings. Prepare notes on your current progress. A meeting with your clients (teacher) will need to be made (outside class time) to discuss various issues. | 2 |
| Total Stage 2 marks | | 25 |
| Total Stage 1, 2 and 3 marks | | 100 |

# Marking Criteria breakdown.

## Stage 1– Understanding the Problem - *Marking Breakdown*

(Software and Course Specifications provided by NESA is the correct guideline that will be used to mark against)

| | | |
|---|---|---|
| Project ideas and decision | • Demonstrate comprehensive understanding of problem, and shows details in different ideas and explain well in chosen solutions. | **nil** |
| • Interface design –storyboard | • Interface design demonstrates that considerable progress has been made from concept sketches/storyboards. Various issues have been considered including layout, S&E issues and functionality.<br>• Interface design demonstrates that some progress has been made from concept sketches/storyboards. A few issues have been considered including layout, S&E issues and functionality.<br>• Interface design demonstrates little progress has been made from concept sketches/storyboards. Only one issue has been considered. | **7**<br>**4-6**<br>**0-3** |
| • Research | • Research demonstrates evidence of multiple sources, both content and delivery (topic and code choice), screenshots are provided with annotation and a clear link between proposed project and research is shown.<br>• Research demonstrates evidence of multiple sources but limited to topic or code. Some screenshots are provided but lack annotation. Link between research and project is vague.<br>• Research demonstrates little evidence of sources. Some screenshots are provided but lack annotation. Link between research and project is not present.<br>• Little or no research is provided. | **6**<br>**4-5**<br>**1-3**<br>**0** |
| Selection of software environment – Reasoning | • Considerable research is evident in the selection of a software environment; including issues related to relevance, skills and time.<br>• Some research is evident in the selection of a software environment; including issues related to relevance, skills and time.<br>• Little or no evidence of selection of the software environment | **2**<br>**1**<br>**0** |
| Identification of hardware/software required | • Hardware and software has been correctly identified for this project with detailed reasoning given for the inclusion of all components.<br>• Hardware and software has been mostly correctly identified for this project with detailed reasoning given for the inclusion of each component.<br>• Some hardware and software has been correctly identified for this project with brief reasoning given for the inclusion of each components or identification is not evident. | **2**<br>**1**<br>**0** |
| • Preliminary Algorithms | • Algorithms are well thought-out and include various modules to be included. Scope of the project is also clear. Algorithms (Flowchart or Pseudo Code) are represented using the correct syntax required for each.<br>• Algorithms are presented with correct structure but are not well thought out and / or scope of project is not evident.<br>• Algorithms are presented with some correct structures but lacks but demonstrates a lack of understanding in the Software Solution. Scope is not determined<br>• Algorithms are poor or do not exist | **4**<br>**3**<br>**1-2**<br>**0** |
| • IPO charts | • IPO charts are accurately presented demonstrating an understanding of all the inputs, processes and outputs of the system.<br>• IPO charts are mostly accurately presented demonstrating an understanding of most of the inputs, processes and outputs of the system.<br>• IPO charts are presented demonstrating an understanding of some of the inputs, processes and outputs of the system.<br>IPO charts are poor or do not exist | **4**<br>**3**<br>**1-2**<br>**0** |
| • Decision tables/trees | • Decision table/tree is accurately constructed demonstrating an understanding of what decisions are to be made within the software solution and what corresponding action will occur.<br>• Decision table/tree is constructed demonstrating some understanding of what decisions are to be made within the software solution and listing some of the | **4**<br>**2-3** |

| | | |
|---|---|---|
| | corresponding action will occur.<br>• Decision tables/trees are poor or do not exist | **0-1** |
| System flowcharts | • System flowcharts are accurately presented demonstrating an understanding of how the Software Solution will interact. Correct syntax has been used.<br>• Correct syntax is used to construct the system flowcharts however understanding of how the Software Solution will interact is not clear or concise.<br>• System flowcharts are poor or do not exist | **4**<br>**2-3**<br>**0-1** |
| Structure charts | • Structure charts are accurately presented demonstrating an accurate understanding of how the Software Solution will interact. Correct syntax has been used.<br>• Structure charts are mostly accurately presented demonstrating a reasonable understanding of how the Software Solution will interact. Correct syntax has been used.<br>• Structure charts are presented demonstrating a small degree of understanding of how the Software Solution will interact. Some correct syntax has been used. Structure charts are poor or do not exist | **4**<br>**2-3**<br>**0-1** |
| Dataflow Diagrams | • Dataflow diagrams are accurately presented demonstrating a clear understanding of the entities, data flows and processes within two levels of the system. Correct symbols and labels have been used.<br>• Dataflow diagrams are mostly accurately presented demonstrating a reasonable understanding of the entities, data flows and processes within two levels of the system. Correct symbols and labels have been used.<br>• Dataflow diagrams are presented demonstrating some understanding of the entities, data flows and processes within two levels of the system. Correct symbols and labels have mostly been used.<br>Dataflow diagrams are poor or do not exist | **4**<br>**3**<br>**1-2**<br>**0** |
| Selection of appropriate data structures (array, record) / Data Dictionary | • Data structures chosen are well considered and appropriate for the processes that is to be carried out. A data dictionary is constructed accurately following the conventions for data dictionary design and variables and constants are appropriately named and defined.<br>• Data structures chosen are poorly considered and not appropriate for the process that is to be carried out. A data dictionary is constructed following some of the conventions for data dictionary design and variables and constants are poorly named and defined OR Data dictionary is designed poorly or is not present.<br>• Data structures are poor or not present. | **4**<br>**2-3**<br>**1-0** |
| • Gantt Chart | • Gantt chart is constructed well with appropriate headings, layout and the time allocations demonstrate understanding.<br>• Gantt chart includes headings, layout and/or the time allocations demonstrate lack of understanding<br>• Gantt chart is constructed poorly with incorrect layouts and headings, inappropriate time allocations. | **3**<br>**2**<br>**0-1** |
| • Developer Diary and Communicate with clients (Log Book) | • Comprehensive diary detailing all developments including the following; progress indicators, research, idea generation, communication, concept sketches and samples. Demonstrates confidence, knowledge and understanding of their software solution, and project management.<br>• The diary details most developments including some of the following; research, idea generation, concept sketches and samples. Conducts the communication process clearly but lacks detail and understanding when considering the software solution.<br>• Diary lacks detail and includes only a few developments such as progress indicators, research, idea generation, concept sketches and samples. | **2**<br>**1**<br>**0** |
| Total Stage 1 marks<br>/50 | | |

## Stage 2– Implementation - Marking Breakdown

(Software and Course Specifications provided by NESA is the correct guideline that will be used to mark against)

| **Implementation and Evaluation** | | |
|---|---|---|
| • Translation of solution into code | • Code is well structured and planned. Translation process from algorithm to code is evident. Code is well commented.<br>• Code is well structured though poorly planned with limited evidence of translation of algorithm to code.<br>• Code is poorly structured with little translation evident. | 9<br>5-8<br>0 -4 |
| • Test data | • Different testing methods have been well researched and discussed. Thorough thought given to why certain data is selected and the test data has been clearly displayed.<br>• Some testing methods have been compiled correctly with minimal thought given to why certain data is selected and test data has been clearly displayed.<br>• Test data has not been considered well or is not present | 3<br>1-2<br>0 |
| • Online help | • Online Help is thorough and comprehensive covering all types of assistance that may be needed.<br>• Online Help is indeed helpful but is not thorough and does not cover all types of assistance that may be required.<br>• Online Help is poor or is does not exist. | 3<br>2<br>0-1 |
| • User Documentation | • Documentation is detailed but user friendly. It includes step by step instructions on performing various tasks and shows evidence of research.<br>• Documentation is detailed but user friendly. It includes step by step instructions on performing various tasks. Research is limited<br>• Documentation is detailed but not as user friendly as it could be. Instructions are clear but follow no real pattern or formation. Research is limited.<br>• Poor documentation or does not exist. | 3<br>2<br>1<br>0 |
| • Developer Diary and Communicate with clients (Log Book) | • Comprehensive diary detailing all developments including the following; progress indicators, research, idea generation, communication, concept sketches and samples. Demonstrates confidence, knowledge and understanding of their software solution, and project management.<br>• The diary details most developments including some of the following; research, idea generation, concept sketches and samples. Conducts the communication process clearly but lacks detail and understanding when considering the software solution.<br>• Diary lacks detail and includes only a few developments such as progress indicators, research, idea generation, concept sketches and samples. | 2<br>1<br>0 |
| • Video Pitch | • Video is well presented, clearly demonstrating what the software does, how it works, future release and why people should invest in it. Use of software screenshots and activity (gameplay) with commentary both written and verbal. Includes consumer prediction and market audience.<br>• Video is well presented, mostly demonstrating what the software does, how it works, future release and why people should invest in it. Use of software screenshots and activity (gameplay) with commentary both written and verbal.<br>• Video is presented with some demonstration of what the software does, how it works, future release and why people should invest in it. Use of software screenshots and activity (gameplay) with commentary both written and verbal are missing or minimal.<br>• Video is not present or shows little promotion. | 5<br>3-4<br>1-2<br>0 |
| Total Stage 2 marks | | 25 |

## Stage 3– Test, Evaluate and Maintain – Final submission. *Marking Breakdown*

(Software and Course Specifications provided by NESA is the correct guideline that will be used to mark against

| | | |
|---|---|---|
| • Maintenance | • Careful consideration has been considered for various maintenance and upgrades.<br>• Some consideration for various maintenance is evident<br>• Poor consideration of maintenance or it does not exist. | 2<br>1<br>0 |
| • Program testing | • Program testing is detailed and includes many methods of testing and the use of appropriate documentation.<br>• Program testing is evident but detailed testing includes only one method of testing and /or lack of documentation presented.<br>• Poor program testing and/or no documentation presented. Testing does not exist. | 4<br>2-3<br>0-1 |
| • Developer Diary | • Comprehensive diary detailing all developments including the following; progress indicators, research, idea generation, communication, concept sketches and samples.<br>• The diary details most developments including some of the following; research, idea generation, concept sketches and samples.<br>• Diary lacks detail and includes only a few developments such as progress indicators, research, idea generation, concept sketches and samples. | 2<br>1<br>0 |
| • Prototype | • Prototype is well constructed and shows evidence of good planning and design. Prototype demonstrates how the product will function once final development has started.<br>• Prototype is well constructed and shows some evidence of good planning and design. Prototype partially demonstrates how the product will function once final development has started.<br>• Prototype is not well constructed and shows little evidence of good planning and design. Prototype partially demonstrates how the product will function once final development has started.<br>• Prototype is poor or does not exist | 15<br>10-14<br>5-9<br>0-4 |
| • Social and Ethical issues | • Social and Ethical issues are addressed appropriately with a demonstration of thorough research and analysis into many issues.<br>• Social and Ethical issues are addressed appropriately with a demonstration of considerable research and analysis into the many issues.<br>• Social and Ethical issues are not addressed or not present. | 2<br>1<br>0 |
| Total Stage 2 marks | | 25 |
| Total Stage 1, 2 and 3 marks | | 100 |

## 4.8 Project Ideas

Stock market simulator
Calculator
Student diary
Study program system
Appointment system
Hotel bookings
Video store management system
Subject choice/report system
Payroll system
Cloze test
Stock control
Batch banking/ audit system
Text encryption/ analysis
Olympic ticketing
Library system
Connect four
Mastermind
Snakes and Ladders
Hangman
Clock Patience
Memory
Battleship
Yahtzee
Draughts
Search a Word
Monopoly
Bingo
Ludo
Guess Who
Cluedo
Sorry
Trouble
Crossword generator
Quizzes
Drill and practice games
Database simulators
Assembler
Sample word processor
Athletics carnival
Resource allocation
Cash register
Tenpin bowling scoring
Calculation of sport averages
Wedding planner

## 4.9 Sample Project

### 4.9.1 Introduction

The following is an example of the type of project and documentation suitable for the student project developed during the Developing Solution Packages unit.
Reasons for selecting such a project include:

- it can be developed in easily identifiable stages
- the student could stop at any of these stages and still have achieved a relevant outcome
- it is interesting for the student to work on, and the interface design is interesting for other students to comment on
- the program code is achievable (for students of higher ability)
- the project includes:
  - file handling (to store and retrieve high scores)
  - arrays (to store current position of each cell of the snake)
  - random numbers (to appear on screen)
  - the need to access manuals (to determine how to draw and move the snake)
  - complex processing (to move the snake around corners).

The project was implemented in incremental stages as outlined in the logbook. Although it is preferable for an overall design to be completed in the initial stages of the program, often this does not happen in the classroom. In this particular program, the teacher and student agreed on this incremental approach to allow each stage to be completed before the next was attempted. Using this approach what started as a fairly basic concept ended up as a complex sophisticated product.

## 4.9.2 Sample Student Logbook

*This logbook is included as an example of the progress of a project from its inception to its completion. It is not intended to demonstrate the best way to implement a project, nor as the best way to format and present a logbook, but rather as an example of a student's progress throughout the year.*

**Week 1**

I decided with the help of my teacher that I would attempt to develop the game Slimy Suckers, in which a snake moves around the screen collecting numbers in order to grow bigger and to finish the level. I had considered a variety of games that I knew, but decided on this one as I had some clear ideas as to how it could be coded, and I knew it would keep me interested!

**Week 2**

The main screens were drawn up on paper including

- welcoming screen
- main menu
- help
- ten levels, with each level having more complicated barriers.

**Week 3**

I received back my screens from my teacher and was advised to only try doing the first level for the moment. Then if I have enough time to go and complete the next couple of levels. I also started to do the documentation for the logic of the program (flow chart). At this point I knew what I wanted to do but had no idea how to do it. I didn't know any drawing functions or how to move an image, and a lot of the things I needed to do weren't straightforward in my mind. After speaking to my teacher and deciding to eliminate the second player, and discovering the INKEY$ function, I found it much easier to understand how I was going to do what I had to.

**Week 4**

I finished off my main algorithm and then started to work with file handling, in order to record my top 10 scores back to disk.

**Week 5**

I started to program the easy bit, the main menu, and all the functions associated with it. I had to consider

- What speed is wanted?
- What colours are to be used?
- How many players wanted?
- What keys do you want to use to move the snake up, down left and right? [Key configuration]

**Week 6**

I took out the Future Basic handbook, and learnt how to use the drawing tools in Future Basic. To my surprise it was quite simple, and I found a couple of functions like a function that made the rectangle move for you, and a function that made the rectangle even grow for you. This I thought would make my life a lot easier. This same week I drew up my opening screen, and started coding the move function.

**Week 7**

I deleted all that I had done the week before, as I realised the functions that I found which would supposedly make my job easier weren't fit for what I needed them for. So, the auto move function, instead of moving the snake, would redraw it, which was a problem because if I was going right and wanted then to move down, my entire snake would rotate, and not 'move'. As a result all that I had done in week 6 was useless to my new plan of action, and that was to do it the hard way. I then started to program the 'move' function into my program, and I incorporated the configuration part of the main menu. The snake by the end of this week could now move one step at a time by command of the user in any direction they wanted.

**Week 8**

I now could get the snake to move, but not how I wanted it to. My snake left a trail of black dots, which wasn't wanted, and there were still problems like I could go left immediately if I was currently going right, making the snake go back on top of itself, which had still to be fixed.

With the help of my teacher, I spent about an hour and a half this week trying to work out why the snake was allowed to move immediately backwards on top of itself. By the end of the afternoon we had put intermediate print statements after every line of code, and had found that, oddly, while setting the dimensions of my snake in a called systems routine, it changed another of my variables. We overcame this problem by assigning this problematic variable with another variable and then reassigning, after the dimensioning of the rectangle, the original variable ie:

```
LET KP = KEEPRESS
CALL SET RECT( RECTG,23,55,67,98)
LET KEYPRESS = KP
```

**Week 9**

I deleted all intermediate print statements. Then I programmed successfully for the snake to move one space at a time, as well as deleting the snake before it. Now it was just one square moving around the screen, yet I could still immediately go left if I was moving right. Once the snake was moving, I had a problem, as the snake was moving too fast for me to control it, or to see what was going on. I looked in the handbook, which told me to use the function DELAY, which would slow my snake down according to the value I put next to the statement.

**Week 10**

Finally I got all the logic working throughout my program, and finally the snake would now not move backwards.

**Week 11**

I started the next stage of my programming and that was to place numbers randomly for the snake to 'eat'. This stage wasn't hard at all, but I had a couple of problems, like where to include the routine, which drew the number, inside the code. I also started coding to validate the keys entered in configuration; it isn't working 100% yet though.

**OVERALL**

I have the logic of the main menu coded but it is not working yet, as I haven't made windows etc. The only function that I have working from the main menu is the configuration of keys. I now finally have my snake moving around the screen (without being able to move backwards) picking up numbers as it goes. As it picks up one number, the next comes onto the screen.

[Logbook handed in at this stage for marking.]

**Week 12**

I started to try to get the snake moving with more than one square at a time, that is with a length of snake. The plan was to keep all coordinates of each following square in an array and then, after the snake moved each time, shift each element of the array, delete the last square, and draw a new square at the front. The array at this stage was to be two-dimensional.

**Week 13**

This week was a bit slow, as I had problems understanding how I was going to do what I wanted to do. This week I still grappled with my problem.

**Week 14**

Again this week was slow, and I realised my main problem was that I had trouble conceptually of how I was going to shift the elements of a two-dimensional array [X coordinate & Y coordinate].

After discussing this with my teacher, she told me that there was a better approach, and that it would be easier to use two one-dimensional arrays. I gave that a try and it worked beautifully!

**Week 15**

This week, after I had got more than one piece of the snake moving at a time, I also now tried to get the snake growing to a certain length. I used a couple of approaches to this problem, for instance a “case where” in which, depending on what number the user was up to, it would grow to a certain length. I finally decided though that the easiest way to do it was to load the corresponding length with the corresponding number into an array from DATA statements. Even though my logic was correct, the function didn't seem to be working, and the variables involved had wacky values for unknown reasons. My teacher and I spent a period trying to find the problem, but unfortunately couldn't.

**Week 16**

A friend heard of my wacky value problem, and came to help me try finding the problem. After searching for two minutes only, he located the problem in my first LONG IF used. The problem was that I used AND statements between my conditions. Now I had my snake growing and moving, but only the first time I played it in one running of the game.

**Week 17**

I now added in a function to recognise if the snake had hit itself as it moved but didn't get it working. After trying for a period I got bored and moved instead onto a validation routine. I wanted to check if the randomly generated coordinates of the next number equalled any of the coordinates of the snake. This didn't take me long, however I couldn't work out how to test my coding, and still haven't been able to.

**Week 18**
This week I tried to iron out a couple of the unfinished functions and processes of my program. I started with the fact that if I played the game a second time, after I had played it once, it didn't work. I added in a subroutine to initialise all my variables, and this helped with some of my problems. I then realised that the problem was that I reloaded the elements of the array used for the specific lengths of the snake after each number, without RESTORING my data statements.

**Week 19**
The last week of term, I cleaned up my code, and added in REM [remark] statements. I also coded the Pause function, and got that working. I also got my Lives working too.

**OVERALL**
This term seemed to be unproductive, as it seems I didn't get that much done. Now however I have my "Game" working apart from the fact that the snake can still hit itself of which the logic has already been coded. As I have already coded the logic of main menu, it shouldn't be long till I fully finish and can get on with help and screen design.

[The Logbook was handed in at this point.]

**Week 20**
This week it hit me how much time I actually had left, and all of a sudden I got a burst of programming power. I started now to instead of putting my problems aside, such as numbers falling on barriers, I decided to tackle them head on, and not procrastinate and not promise that I'd do them later. This week I thought that I finally solved the problem of numbers landing on the snake or other barriers.

**Week 21**
It was the first week of holidays, and I had the laptop. Because I couldn't run the game on the laptop, I decided that I was going to start to now code properly the main menu. Up until now, I had the logic of the main menu in another file, but I as yet had no windows, print or input statements, only the logic. It was this week that I finished coding the logic of the main menu, and mentally made links between the menu and the game. I couldn't do any screen design, though, as I couldn't see colours on the black and white laptop.

**Week 22**
This week was the second week of the holidays, and again I had the laptop for the day. During the day I started to try to do screen design on the laptop, but this proved fruitless. However since I couldn't run my game on the laptop, I decided to keep trying with the main menu and in particular the screen design. By night I hadn't done much, though I had achieved a little bit more than what I already had, and if nothing else, I worked out in my head what the layout of each screen would look like. That night, however, before I went to bed, the idea of having different levels formed in my head. I originally thought this would be impossible, and should only be left to the very end. The next morning though with nothing to lose, I started to code the idea of different levels into the game. I couldn't however see if what I was doing was correct, as the program wouldn't run.

**Week 23**

This week we were back at school, and I could finally see properly if what I had done during the holidays was worth my time, and if it was working. Surprisingly the levels that I tried to code were very close to correct, and with a little bit of tinkering with the code they were working. This, I thought, was a great achievement. However, with the introduction of the new levels came more complications:

1) I now had to not only draw the snake, and the border, but now had to work out how to draw the obstacles.
2) Not only did I have to watch if the snake hit itself or the walls, but now I also had to watch if it hit any of the obstacles.
3) I originally had problems with the numbers appearing on the snake and borders; I now also had to worry about the numbers landing on the obstacles.

These new challengers made the code very much more complex, and it caused a chain of problems. As I started to fix each of the problems during this week, I seemed to create more problems.

I now had to worry much more about the order of each of the processes, for instance does the program work if I check if the snake has hit itself before I check to see if the snake has hit a barrier. It is hard to believe, but this 'order of subroutines' took me at least a whole day to figure out.

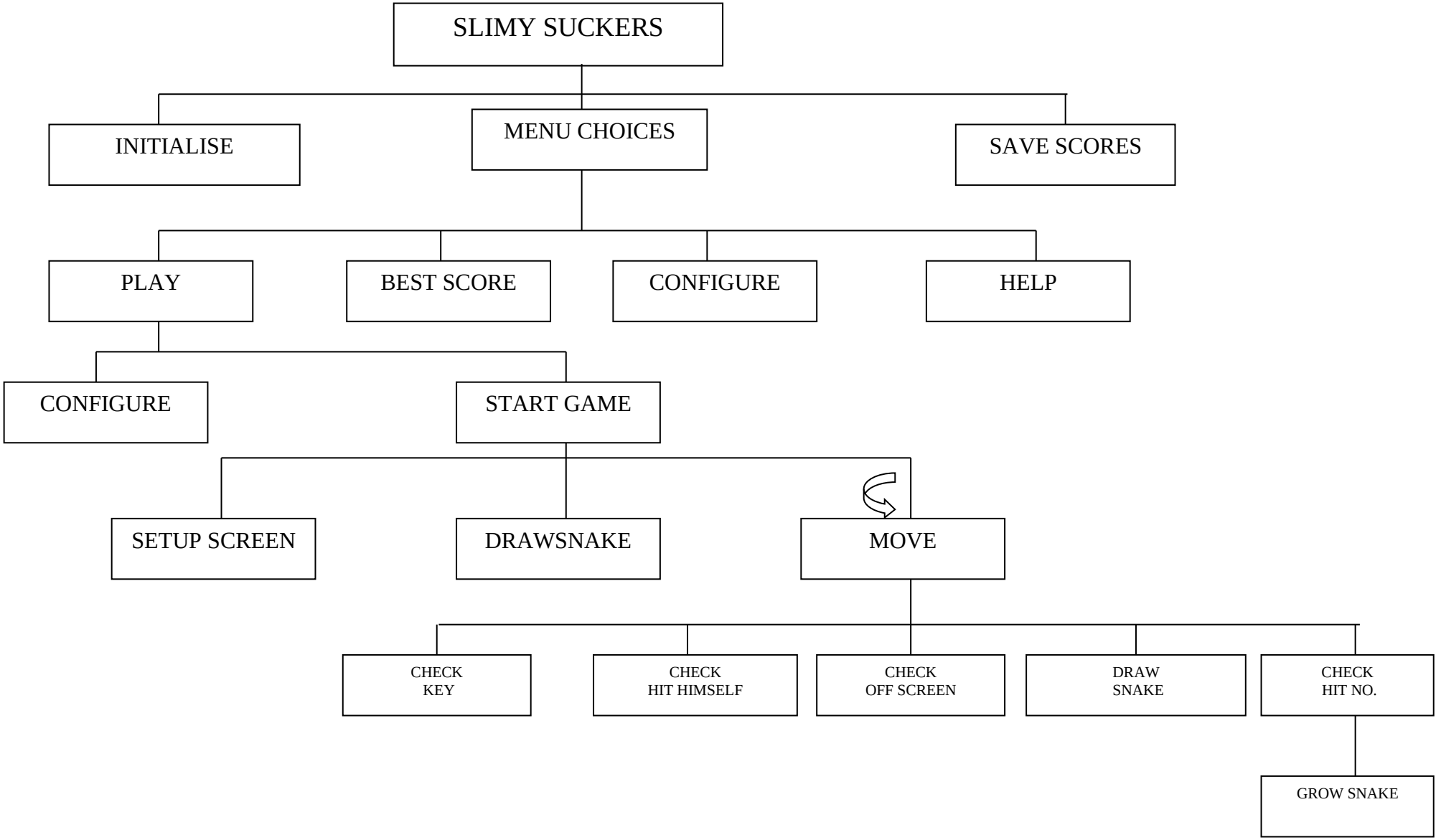
**Week 24**

I continued to struggle with the introduction of the levels, but at the same time moved on. I started to combine the main menu with the game, making sure there were no double variables, and that the relevant and correct variables used in the main menu linked with that of the game. For instance, the choice of speed entered in the main menu had to be used in the "Speed" subroutine in the game. I also had duplicates of some routines which had to be deleted or integrated, for instance the configuration of keys. This week also with the help of a friend, I managed to use a mouse to pick the colours of the snake and the border of the playing field. I also added the option of choosing the colour background, whether the player wanted it black or white. This, though, caused another small problem in that now on a black background the snake would delete the background, and leave a white trail. To overcome it, instead of deleting the last square, I drew either a white or black square over the last square, depending on the colour of the background.

**Week 25**

This week to my surprise I completely finished my program, with all of the logic working! I started with screen design, and a friend taught me how to use Res-edit, as I didn't have time to learn it myself. It was actually quite easy. I drew my opening screen and after realising that it was hard to distinguish between the menu options and the questions, I decide to highlight the first letter of the question. And so my screen design was born. I continued to follow suit with each of the subsequent screens. This took a long time, as I had to basically guess and check the coordinates of the start of each sentence, as well as the position of the inputs. This process took me most of the week. On the last night, I managed to do some screen design for the actual game, with messages popping up inside boxes, instead of in corners of the screen. I also managed to add in error messages; before this, even though inputs were validated, they never said what the problem was. I also coded the help menu in the last night, and ironed out all errors.

### 4.9.3 Structure Chart



### 4.9.4 Data Dictionary

| VARIABLE | LENGTH | TYPE | DESCRIPTION |
|---|---|---|---|
| again$ | 1 | Alpha | Asks the player whether he'd like to play again |
| back | | numeric | variable name of the square dimensioned (back square) |
| backrou$ | 1 | Alpha | colour of the background |
| backrou | | numeric | colour of the background |
| colourl | 10 | Alpha | colour of the border |
| colour2 | 10 | Alpha | colour of the snake |
| cover | | numeric | used so the next number doesn't appear where the snake is located |
| falg | | numeric | a flag set to know whether to increment the variable 'numb' |
| flag | | numeric | program will drop out of the 'move' loop to perform a different task |
| helpch | | numeric | choice from help menu |
| increase$ | 1 | Alpha | whether the speed increases through the game |
| index | | numeric | An index used in "Configuration'[9] to be index to direction$(5) |
| keypress | | numeric | the ACSII value of keypress 1$ |
| keypress 1$ | 1 | Alpha | the key entered by the player to make his snake move |
| lastkeypress | | numeric | before a new keypress is entered, the old one becomes lastkeypress |
| Lives | | numeric | How many lives the player has left |
| mmenu | | numeric | choice from main menu |
| Nam$ | 15 | Alpha | name of player |
| Nexts | | numeric | The level number that the player is up to |
| Numb | | numeric | The counter of which is the next number the snake must 'swallow' |
| pause | | numeric | flag set to say the game was just paused |
| rectg | | numeric | variable name of the square dimensioned (front square) |
| rect | | numeric | variable name of the square dimensioned (front square) |
| Rndmx | | numeric | a random horizontal value where a number will be placed to be collected |
| Rndmy | | numeric | a random vertical value where a number will be placed to be collected |
| Shape | | numeric | The choice of shape by the user, It determines the shape of the snake |
| score! | | numeric | the players score throughout the game |
| snake$ | 20 | Alpha | name of player's snake |
| speed | | numeric | speed at which the snake moves |
| pole | | numeric | a flag set to tell if the validation of the keys was OK |
| upto | | numeric | A count to see what part of the subroutine it is up to |
| vpos | | numeric | the vertical position of an error message |
| hpos | | numeric | the horizontal position of an error message |
| x | | numeric | the current position of the snake (horizontal) |
| Y | | numeric | the current position of the snake (vertical) |
| **ARRAYS** | | | |
| bestname$ | 10 | array | array where top ten players are sorted and then written to disk |
| bestscores | 10 | array | array where top ten scores are sorted and then written to disk |
| direction | 5 | array | these keys are converted to ASCII values |
| direction$ | 5 | array | player chooses the keys he wishes to play with |
| done | 10 | array | an array pointing out if a message has already been done |
| startx | 640, 200 | array | The starting "x" position of the barrier and the number barrier |
| Starty | 480, 200 | array | The starting "Y" position of the barrier and the number barrier |
| endx | 640, 200 | array | The end "x" position of the barrier and the number barrier |
| endy | 480, 200 | array | The end "y" position of the barrier and the number barrier |
| snake$ | 640, 480 | array | Horizontal and Vertical Position of the snake |
| Lengt | 9 | array | As each number is picked up, the snake grows to this certain length |
| Randm | 10 | array | an array containing 10 different random messages |

### 4.9.5 Gantt Chart

| **Slimy Suckers** | | | April | | | | May | | | | | June | | | | July | | | | August | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | 22 | 29 | 5 | 12 | 19 | 26 | 3 | 10 | 17 | 24 | 31 | 7 | 14 | 21 | 28 | 5 | 12 | 19 | 26 | 2 | 9 | 16 | 23 |
| learn to draw | | | | | | | | | | | | | | | | | | | | | | | |
| get snake moving | | | | | | | | | | | | | | | | | | | | | | | |
| add in numbers | | | | | | | | | | | | | | | | | | | | | | | |
| grow the snake | | | | | | | | | | | | | | | | | | | | | | | |
| check for numbers on barriers | | | | | | | | | | | | | | | | | | | | | | | |
| check if numbers on snake | | | | | | | | | | | | | | | | | | | | | | | |
| check if snake hits itself | | | | | | | | | | | | | | | | | | | | | | | |
| add in extra levels | | | | | | | | | | | | | | | | | | | | | | | |
| fix all logic! | | | | | | | | | | | | | | | | | | | | | | | |
| interface polishing | | | | | | | | | | | | | | | | | | | | | | | |
| online help | | | | | | | | | | | | | | | | | | | | | | | |
| documentation | | | | | | | | | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | | | | | | | | |

### 4.9.6 Code Fragments

These code fragments are written using FutureBasic (a version of BASIC that runs on Macintosh computers)

Specific routines have been selected to indicate the type of code that can be expected at this level

**1. The printing of random messages to the user**

```
‘sets the duplicates array to zeros, to prevent a random number being picked twice
LOCAL FN initialisemsg
        i = 1
        DO
                done(i) = 0
                i=i+1
        UNTIL i = 9
        count = 0
 END FN

'reads messages from data statements
LOCAL FN readmssg
        i= 1
        WHILE <=10
                READ randm$(i)
                j=j+1
        WEND
END FN

'prints random messages by selecting a random number that has not already been picked
LOCAL FN message
        DO
                randm = INT(RND(10))
        UNTIL done(randm) = 0
        PRINT randm$(randm)
        done(randm) = 1
        count = count +1
        IF count = 8 THEN
                FN initialisemsg
        END IF
END FN


DATA Sing for your salvation as you slide into the slippery realm of the Slimy Suckers
DATA Surging energy may safeguard those who salute the severe Slimy Suckers
```

DATA Sanctify the sacred serpent solemnly with a sacrifice only then will you escape the wrath of the Slimy Suckers
DATA The saliva's smell will even spook those who are squeamish beware the stench of the Slimy Suckers
DATA See if you can sense the stealth of the snake as it sneaks slowly to satisfy its purpose as a Slimy Sucker
DATA Serve your senior or its strength with strangle and squish too strong are the Slimy Suckers
DATA Stupefy and sorry yourself in order to snatch sovereignty from the skilful Slimy Suckers
DATA She sells sea shells by the sea shore they eat sea lions on the sea shore they are the Slimy Suckers

## 2. Check to determine direction of movement, and not allow the snake to move backwards over itself

```
'direction of movement
LOCAL FN moveup
        LONG IF lastkeypress <> direction(2)
                y = y -10
        XELSE
                keypress = direction(2)
        END IF
END FN

LOCAL FN movedown
        LONG IF lastkeypress <> direction(1)
                y =y + 10
        XELSE
                keypress = direction(1)
        END IF
END FN

LOCAL FN moveleft

        LONG IF lastkeypress <> direction(4)
                x =x - 10
        XELSE
                keypress = direction(4)
        END IF
END FN

LOCAL FN moveright
        LONG IF lastkeypress <> direction(3)
                x =x + 10
        XELSE
                keypress = direction(3)
        END IF
END FN

'the move function, to move the snake according to keys pressed by the user.
'These keys have previously been selected by the user to indicate up, down, right and left
LOCAL FN move
        DO
                'print 9 random numbers…
                WHILE falg = 1 AND numb < 10
                        IF numb <=9 THEN FN ranmno.
```

```
                numb = numb + 1
                FN score
                falg = 0
        WEND

        'get a keypress from the user
        WHILE pause = 0
                keypressl$ =INKEY$
                keypress = ASC(keypressl$)
                IF keypressl$ = "q" THEN STOP
                IF keypress = 0 THEN keypress = lastkeypress
                pause = 1
        WEND

        'work out where to move next
        pause = 0
        SELECT keypress
                CASE direction(1)
                        FN moveup
                        lastkeypress = keypress
                CASE direction(2)
                        FN movedown
                        lastkeypress = keypress
                CASE direction(3)
                        FN moveleft
                        lastkeypress = keypress
                CASE direction(4)
                        FN moveright
                        lastkeypress = keypress
                CASE direction(5)
                        FN pause
        END SELECT

        FN checkhit         'now check to see if the snake has crossed over itself
        LONG IF flag <> 2
                        FN shift
                        FN drawsnake
        END IF
        FN speed            'wait for the user specified delay to slow the snake down
        FN checkplace       'now check to see if the snake has run over a number
    UNTIL flag = 2  OR numb = 11
END FN
```

## 3. Prints the random number at a position that is ok (ie not under the snake!)

```
'prints a number (generated earlier and stored in the variable numb) in a random position
LOCAL FN ranmno.
falg = 0
'loop to test if the number will print in co-ordinates taken up by the snake
t=1
DO
    PRINT
    l =1
    cover = 0
    i = 1
    rndml =     INT(RND(57)) + 1
    rndm2=      INT(RND(35)) + 4
    radmx= rndml*10
```

```
        radmy= rndm2*10
        WHILE I =< length(numb - 1) AND cover = 0
                LONG    IF radmx = x(i) AND radmy = y(i)
                        cover = 1
                        XELSE
                          i = i +1
                        END IF
        WEND
        LONG IF nexts> 1
                        WHILE I <= amount(nexts) AND cover = 0
                            LONG IF radmx >= startx(nexts,I) + 10 AND radmx < endx(nexts,I) +
10
                            AND radmy >= starty(nexts,I)+10 AND radmy < endy(nexts,I) + 10
                            cover = 1
                            END IF
                            I  = I +1
                        WEND
        END IF
UNTIL cover = 0
'checks the colour of the background to make sure the number can be seen
IF backrou$ = “w” THEN
        TEXT _geneva, 10,0,_srcCopy
ELSE
        TEXT _geneva, 10,0,_nosrcCopy
END IF
PRINT %(radmx,radmy) ;numb
END FN
```

### 4.9.7 Installation Guide

# Installation Guide

Congratulations, you are the lucky owner of the new game Slimy Suckers™

## 1) Equipment Needed

To play Slimy Suckers, you will need a Macintosh computer — nearly any one will do, but you will find that the game will run best on a Macintosh that is more recent, for instance a LC575. The computer selected will also need to have access to Future Basic – II®.

In the purchased box you should find, along with this Installation Guide and Tutorial, a 3.5 inch floppy disk (the small grey square). Once you have identified an appropriate PC you will have to insert this floppy disk into the floppy disk drive, that is the small slit found at the base of the screen.

## 2) Getting into the program

Once the disk is inserted, you will notice after a few seconds that a small icon in the shape of the disk you just inserted into the machine will appear somewhere on the right-hand side of your screen. The caption under this icon will read Slimy Suckers.

If you're having trouble so far, don't worry, from here it is all downhill.

The next step is to simply position the mouse over the icon labelled Slimy Suckers, and double-click. You should see a window (box) appear in the center of your screen. If for some reason this event doesn't occur, it is probably because you haven't double-clicked fast enough. You'll have to try again. Now get cracking! (By the way, if you can't double-click fast enough, I'd hate to see how you'd play the game.)

In the window that has just been opened you should be able to find another icon labelled "Slimy Suckers". You'll have to again double-click on this to enter the program. If you can't find the icon try using the scrollbars to the right and bottom of the window to locate it.

## Starting the game

Now, you have finally reached the light at the end of the tunnel.
To start playing the game, all you have to do, is either:

- Simultaneously press the “Apple” key and the letter “R” on the keyboard
- Or go up to the top of the screen and click on the heading and then move down the menu to “Run” and click again.

Now you are finally in the program. You should notice that another window comes up labelled at the top “Open”. At this step you are opening the top ten players to date. Using the scroll bar on the right of the small window found in figure 1.3 you have to locate the file named “BEST”. Once you have located the file, you can either double-click on it, or select it by clicking on it once with your mouse, and then clicking on the oblong shaped “Open” button as seen in figure 1.4.

Congratulations, you have just completed the installation process for Slimy Suckers™.

### 4.9.8 Tutorial

After completing the installation, you should now be ready for the tutorial.

# Tutorial

## Table of Contents

### Introduction



### Start Game



# Introduction

## 1) Main Menu

At the main menu (figure 1.1) you have a few options. You can:

1. Start a new game
2. View the highest scores to date (Slimiest Suckers)
3. Configure playing keys
4. Ask for help
5. Quit.

To make a selection, simply press the number of the choice you would like, and press enter.

Figure 1.1


## 2) View highest scores

Viewing the highest scores, is an option which allows one to view who has got the highest score so far, and what that score is.

To view the highest scores, press the number “2” in the main menu (seen in figure 1.1) screen and then press “Enter”.

A screen (figure 1.2) will open containing a list of all the high scores along with the person, and name of their pet snake.

To exit this screen just press “Enter”  and key.

When the number “2” is entered into the main menu a screen such as this should be revealed

Figure 1.2

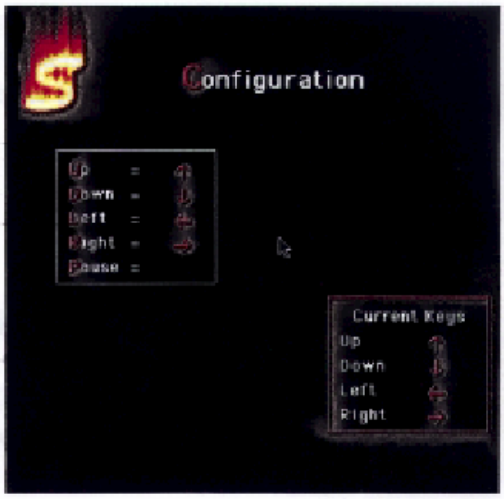
## 3) Configure Playing keys

In this step you will enter preferred keys for use as Up, Down, Pause, Help etc. while playing Slimy Suckers.

To access the “configure keys” function, you just have to enter “3’ into the main menu (figure 1.1). A screen will come up entitled “Configuration”.

To configure your playing keys simply enter in each key (as figure 1.3) according to the direction/function that you are up to. As you get to the next direction, “=” will print on the screen, and from this you can determine which direction/function you are up to.

Figure 1.3




The default settings for the keys are as follows:

- Up = ↑
- Down =↓
- Left = ←
- Right = →
- Pause = p

These settings will be used every time the program is opened. However, once you have entered your own key assignments, they will be used for the rest of the time you play the game.

NOTE: You do not have to press Enter after entering each key. The “cursor” will move to the next direction as soon as any key is hit.

## 3) User Help

User help provides information about how to play Slimy Suckers, and also about the nature of the game.

To gain access to the user help menu, you must type “4” in the main menu, (figure 1.1). A screen (ie figure 1.4) will print on the screen.

This screen gives you three options. The first option is to find out what the game Slimy Suckers is, and the second option instructs the user on the purpose of the game, as well as explaining how to play it.

The third option is simply to return back to the main menu.

## 4) Exit

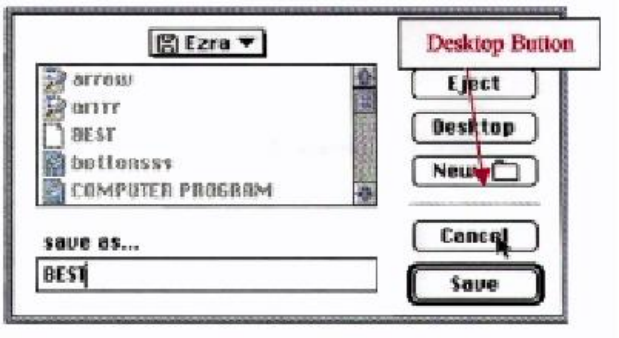
This function is to leave the game in its entirety.

After you select “5” from the main menu, the game will end. Before ending, however, the new (and hopefully improved) “Slimiest Suckers” must be saved

to the 3.5 inch floppy disk. On quitting, a window will be revealed (Figure 1.4), requiring something to be typed.

In the text box the word “BEST” must be entered and then you have to click (using the mouse) on the button “Desktop”, and then click again in the display box on the icon labelled “Slimy Suckers”. Finally the last step before exiting is to click on the button “Save” to save the highest scores to disk.

Figure 1.4




# Playing the Game

## 1) Start the Game

Enter “1” in the main menu (figure 1.1).

Before the game starts, you will be asked to enter different types of information concerning game options and your own details.

The first of these is a screen entitled “Details”.

### a) Details

In this part of the program, you will be asked to enter your name, and the name of your pet snake, for use in case you make your way into the “Slimiest Suckers”.

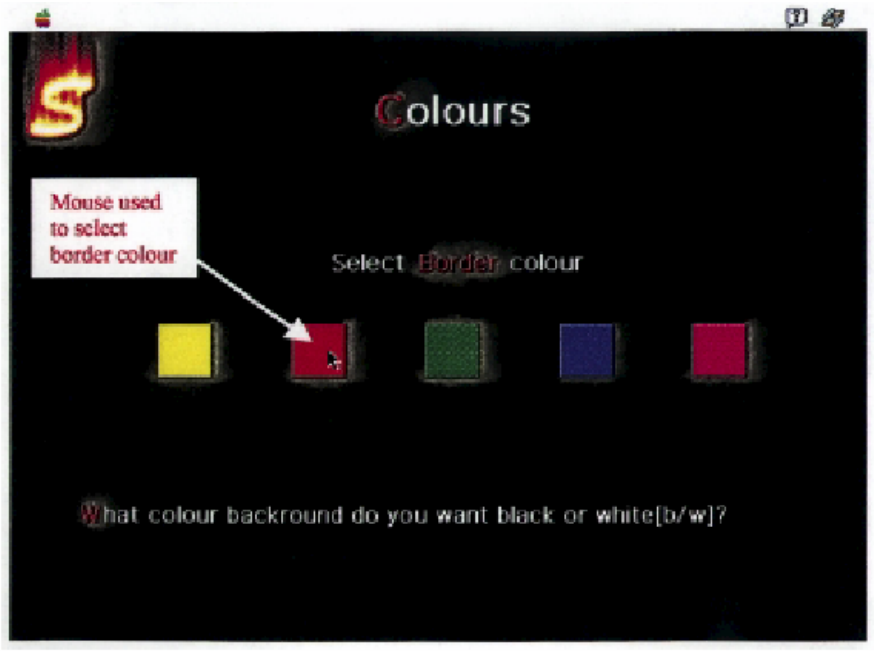
Note: The names may not be more that 15 letters long. If they are an error box will appear, and you will have to re-enter your, or your snake’s name.

In this process you also have the option of changing any of your names. If you don’t want to, simply type the letter “n”.

### b) Colours

After entering your name and your pet snake’s name, you are presented with colour options (figure 2.2).

Figure 2.2




You are asked for your choice of colour for your snake, and also for your choice of colour for the borders of the game. To make your choice you must use the mouse, and must click on the preferred colour.

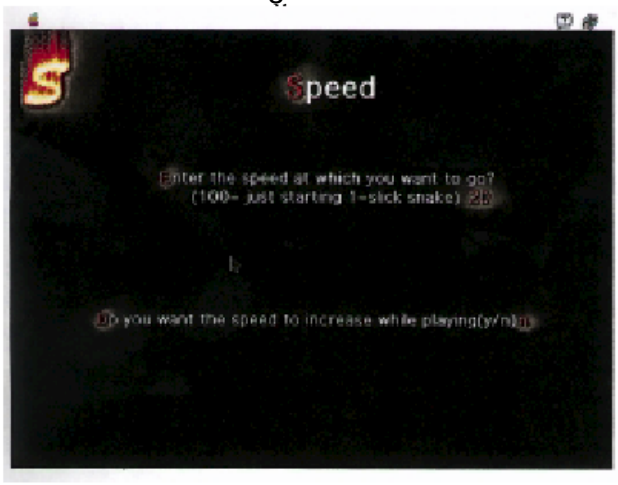
After selecting which colour snake and border you would like, you are given the option of a white or black background. To make your choice simply type either “b” for black or “w” for white.

**c) Speed**

This next screen is for you to enter your most comfortable playing speed (figure 2.3).

100 is the slowest speed (just starting) while 1 is the fastest speed (Slick Snake).

Figure 2.3



You are also asked if you would like the speed to increase while playing. What this means is that at every level that you pass, the speed will increase by 2.

**d) Shape**

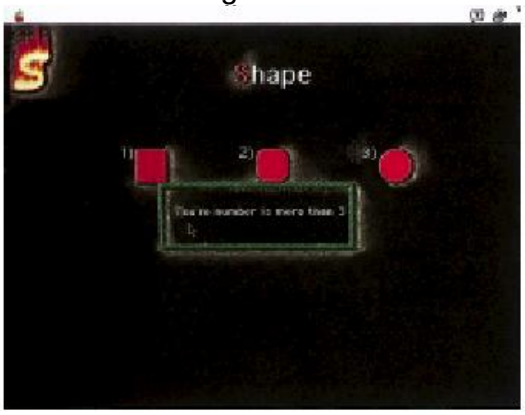
The final option before playing the game is to select the shape of the snake to be used.

The option presented on the screen (figure 2.4), requires the selection to be keyed in, and not to be entered using the mouse (although it may seem like it).

Figure 2.4


The option entered must lie between 1 and 3 for it to be valid. If the user does not enter a correct value, an error message will appear (fgure 2.5), and the user will have to re-enter a valid value.

Figure 2.5




Finally the game can begin. Use your configured keys to move the snake around the screen to collect numbers. Enjoy.