Software Design and Development
Stage 6
Support Document
1999
## Contents

1 Introduction ........................................................................................................... 4

2 Information Specific to the Units of Work ......................................................... 4
   2.1 Resources........................................................................................................... 4
   2.2 Unit Length...................................................................................................... 4

3 Programmed Units of Work ............................................................................. 5
   3.1 Preliminary Course: Software Development Approaches ............................ 5
      3.1.1 Task Specification ..................................................................................... 9
      3.1.2 Approach Summary Sheet ...................................................................... 10
   3.2 HSC Course: Developing a Solution Package ............................................. 11

4 Managing and Assessing a Project................................................................. 21
   4.1 Purpose of Project Management............................................................... 21
   4.2 Tools for Project Management ................................................................. 21
      4.2.1 Gantt Charts .......................................................................................... 21
      4.2.2 Logbooks .............................................................................................. 21
   4.3 Steps in Project Management ..................................................................... 22
   4.4 Teaching Strategies for Group Work ......................................................... 22
   4.5 Group Projects ............................................................................................ 22
      4.5.1 Forming student work groups ............................................................... 22
      4.5.2 Roles of individual students .................................................................. 23
   4.6 Monitoring of Student Progress (individual/group) .................................... 23
   4.7 Assessment (individual and group) ............................................................. 23
   4.8 Project Ideas .................................................................................................. 24
   4.9 Sample Project ............................................................................................. 24
      4.9.1 Introduction ............................................................................................ 24
      4.9.2 Sample Student Logbook ...................................................................... 25
      4.9.3 Structure Chart ...................................................................................... 30
      4.9.4 Data Dictionary ...................................................................................... 31
      4.9.5 Gantt Chart ............................................................................................ 32
      4.9.6 Code Fragments ..................................................................................... 33
      4.9.7 Installation Guide ................................................................................... 36
      4.9.8 Tutorial .................................................................................................. 37

5 Preliminary Assessment Scheme ..................................................................... 44
   5.1 Example .......................................................................................................... 44
   5.2 Task Outlines ................................................................................................. 45

6 HSC Assessment Scheme ................................................................................. 46
   6.1 Example .......................................................................................................... 46
   6.2 Task Outlines ................................................................................................. 47
1 Introduction

This support document is designed to assist teachers as they plan for the implementation of the Software Design and Development Stage 6 Syllabus.

This support document provides programming and assessment ideas for selected syllabus content. Each unit of work relates to an area of study; however, teachers may elect to teach the areas of study in an integrated manner.

Resources related to each unit of work are included. However, it should be noted that a more extensive list of subject specific resources is provided on the Board of Studies website http://www.boardofstudies.nsw.edu.au

2 Information Specific to the Units of Work

2.1 Resources
Each unit of work has a variety of resources listed. The intention is that teachers may select from the list provided to assist in the delivery of the unit.

2.2 Unit Length
A suggested unit length has been provided; however, teachers may elect to alter this. In some cases, certain aspects of a unit can be integrated or combined.
3 Programmed Units of Work

3.1 Preliminary Course: Software Development Approaches

Suggested Time Allowed:

Four weeks as a stand-alone unit. If this unit is integrated with Social and Ethical Issues and/or Developing Software Solutions, then more time should be allocated.

Rationale:

The aim of this unit of work is to describe and differentiate between the four syllabus approaches to software development. It is important that the similarities and differences between these approaches be addressed explicitly at the beginning of the unit and again at the end of the unit.

The Ticketing Project

The Ticketing Project is developed to illustrate each of the four approaches to software development described in the syllabus:

1) structured
2) prototyping
3) rapid applications
4) end user.

In implementing all four approaches for the one scenario, students begin to understand the complexity of a single problem, rather than having to come to terms with the requirements and logical constructs needed to solve four separate problems.

Ticketing can be set in any scene that the teacher feels will establish relevance for their particular student population, such as a football game, the school play or a rock concert. The essential elements require a system to:

(a) collect and account for ticket sales money;
(b) determine ticket prices based on seat location, concession and discount rules, and print the tickets;
(c) allow the public to calculate the price of their tickets online.

The end user approach — applying application software in its simplest and most direct form — will be used to implement a system to collect and account for ticket sales.

The rapid application approach — using scripts and macros from within the application software — will be used to develop the system for determining ticket prices based on seat location, concession and discount rules and to print the tickets.

Prototyping — enabling the user to evaluate the screen designs, button options and links — will be used to implement a system to allow the public to calculate the price of their tickets online.
Finally, when the students have developed a strong understanding of the requirements and logical constructs of the ticketing problem, the *structured approach* can be introduced, requiring the student to develop algorithms to encompass the entire problem, and to code relevant parts of the problem in a structured manner.

As extension material for each approach, the students could be required to modify their solution to ensure that the system could be used for future events, to set up data structures for an event running over more than one day and to provide reports on seat occupancy.

**Resources:**

A spreadsheet, a relational database, an event-driven language, a structured procedural language, CASE tools and any stimulus material suited to the ticketing scenario such as tickets, seating plans, reports and examples of screen layout.

**Texts**


**Possible Assessment Strategies:**

Assessment for each phase of the project requires evaluation of the developed software and the documentation associated with that software. Students should also submit a report on the approach used in each phase. The report should describe the advantages and disadvantages of the approach and an evaluation of it based on their experiences with the Ticketing Project. See the suggested table layout in section 3.1.2 which should be filled in progressively as each task is completed.
<table>
<thead>
<tr>
<th>Outcomes</th>
<th>Students learn about:</th>
<th>Students learn to:</th>
<th>Strategies, activities and related resources:</th>
</tr>
</thead>
<tbody>
<tr>
<td>A student:</td>
<td><strong>End user approach to software development</strong>&lt;br&gt;- characteristics of end user approach, including:&lt;br&gt;  - use of standard software packages&lt;br&gt;  - lack of formal stages&lt;br&gt;  - short time period&lt;br&gt;  - potential long-term, small-scale project&lt;br&gt;  - low budgets&lt;br&gt;  - end user is the developer</td>
<td><strong>select appropriate software development approaches for specific purposes</strong></td>
<td>Students:&lt;br&gt;  - produce a solution to a ticketing problem using a spreadsheet package (no macros) to manage a ticket selling project&lt;br&gt;  - identify input and output needs to meet teacher set specifications and include project plans and user documentation&lt;br&gt;&lt;br&gt;An example of this task is given in section 3.1.1</td>
</tr>
<tr>
<td>P2.2</td>
<td>explains the effects of historical developments on current practices</td>
<td><strong>The rapid applications software development approach</strong>&lt;br&gt;- characteristics of the rapid approach, including:&lt;br&gt;  - lack of formal stages&lt;br&gt;  - coding languages used&lt;br&gt;  - relationship of programmer to end user&lt;br&gt;  - short time period&lt;br&gt;  - small-scale projects&lt;br&gt;  - low budgets&lt;br&gt;  - involvement of personnel, including developer and end user</td>
<td>Students:&lt;br&gt;  - use an existing software package to develop a customised solution&lt;br&gt;&lt;br&gt;An example of this task is given in section 3.1.1</td>
</tr>
<tr>
<td>P3.1</td>
<td>identifies the issues relating to the use of software solutions</td>
<td></td>
<td></td>
</tr>
<tr>
<td>P4.1</td>
<td>analyses a given problem in order to generate a computer-based solution</td>
<td></td>
<td></td>
</tr>
<tr>
<td>P4.2</td>
<td>investigates a structured approach in the design and implementation of a software solution</td>
<td></td>
<td></td>
</tr>
<tr>
<td>P4.3</td>
<td>uses a variety of development approaches to generate software solutions and distinguishes between these approaches</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Outcomes A student:</td>
<td>Students learn about:</td>
<td>Students learn to:</td>
<td>Strategies, activities and related resources:</td>
</tr>
<tr>
<td>---------------------</td>
<td>-----------------------</td>
<td>---------------------</td>
<td>------------------------------------------------</td>
</tr>
</tbody>
</table>
| P6.1 describes the role of personnel involved in software developments | **The prototyping approach to software solutions**  
- characteristics of the prototyping approach, including:  
  - non-formal  
  - shorter time period  
  - small-scale projects  
  - small budgets  
- involvement of personnel, including programmer and users  
- links with structured approach | **design and develop a limited prototype as a demonstration of a solution to a specified problem** | Students  
- will use a prototyping approach to develop a software solution providing a user-friendly event-driven package that will enable clients to test various ticket combinations to identify the best price  

*An example of this task is given in section 3.1.1* |
| | **The structured approach to software solutions**  
- program development cycle for structured approach, including defining the problem, planning, building, checking and modifying  
- characteristics of structured approach, including:  
  - long time periods  
  - large-scale projects  
  - large budgets  
- involvement of personnel, including analysts, designers, programmers, users and management  
- team approach | **identify each of these stages in practical programming exercises** | Students  
- will design a structured solution and produce one overall program, which takes into account all necessary elements for the ticketing project. The solutions should include  
  - the ability to store, track and produce tickets, and  
  - provide constant feedback as to ticket availability and/or sales reports  

*An example of this task is given in section 3.1.1* |
3.1.1 Task Specification

- **End User Approach**
  
  **Specifications**
  
  A theatrical group wishes to run a stage show and to do this they have to sell tickets.

  Your task is to provide a spreadsheet solution that will enable the theatre management to see, at any stage, how many tickets have been sold, how many have been collected, how many are still available and to see a running total of funds. The theatre managers should also be able to identify how many tickets are allocated to each buyer, the buyer’s name and contact number and the rate(s) at which the tickets were sold to the buyer (ie adult or concession or family/group rate).

- **Rapid Application Approach**
  
  **Specifications**
  
  The theatrical group wants to be able to manage seat allocation and funds. The application, developed in a database or spreadsheet together with scripts and macros, should be able, with the press of a button, to prepare the tickets and print them on plain paper showing all relevant details. Further macros under buttons should enable specific sorts, searches and links to graphs of the ticket sales. Reports should be able to be generated showing the proportions of each type of ticket sold.

- **Prototyping Approach**
  
  **Specifications**
  
  A program is to be generated that will enable customers to decide on the appropriate ticket package or combination of tickets which suits them best. Simple mouse clicks or keyboard entries on a series of guided self-help screens will enable them to cost packages and try alternatives.

  Screen layouts and flows of information between screens will have to be documented as well as the processing that will occur in the background of each page. Instructions are to be intrinsic to the screens.

  The code could be achieved in HyperCard, Visual Basic, Delphi etc.

- **Structured Approach**
  
  **Specifications**
  
  A menu-driven package is to be developed that will enable a ticket seller to enter customer details and set numbers of tickets. Ticket information will be stored and the program will both prepare the tickets for printing, and provide reports of tickets sold according to type and ticket availability. Arrays may be used to identify location of seats and simple text output could be used to show either location of seats allocated to a customer or remaining available seats.

  **NOTE:** It is expected that the development of this solution will be followed right through in algorithm form; however, the choice of modules to be coded is at the discretion of the teacher.

  Suitable programming languages Visual Basic, Delphi, QBASIC, Pascal etc.
### 3.1.2 Approach Summary Sheet

<table>
<thead>
<tr>
<th></th>
<th>End User</th>
<th>Rapid Application Development</th>
<th>Prototyping</th>
<th>Structured Programming</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Features</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Applications</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Advantages</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Disadvantages</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
3.2 HSC Course: Developing a Solution Package

Suggested Time Allowed:

Six months elapsed time, from the start of term one with the submission of the final project in July/August.

Rationale:

In this unit, students are required to complete a relevant individual project, incorporating the design and development of a substantial piece of software. It is suggested that between a quarter and a half of the total course time be allocated to the project. This offers students the opportunity to put into practice many of the learn to requirements of the syllabus, at the same time as they experience development of a project.

Throughout the project, students will develop skills in communication and project management, as well as skills more specifically required for the design and development of computer software. They learn to develop software that will meet the needs of a range of potential users, as they are encouraged to both offer and accept feedback from their peers and the teacher.

Resources:

*Project Development Environment*
Access to one (1) computer per student
Access to language reference manuals and help files
Internet access to FAQs and newsgroups for the selected language
CASE software or equivalent, and the relevant documentation
Backup medium
Access to colour printer
Library routines

*Texts*

Possible Assessment Strategies:
Formal assessment is based on the logbook, system documentation, source code, interface design, and management of the project at two (2) stages during the project (end of term 1 and final submission).
<table>
<thead>
<tr>
<th>Outcomes A student:</th>
<th>Students learn about:</th>
<th>Students learn to:</th>
<th>Strategies, Activities and Related Resources:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Designing and developing a software solution to a complex problem</td>
<td>Defining the problem and its solution including: • defining the problem – identification of the problem – idea generation – communication with others involved in the proposed system</td>
<td>This topic starts at the beginning and continues until the end of the HSC course. It is suggested that some periods every week be allocated to the ongoing project while the rest of the week is dedicated to teaching associated theory. Towards the end of the project teachers might find that they need to allocate more class time for project work.</td>
</tr>
<tr>
<td>H1.1 explains the interrelationship between hardware and software</td>
<td></td>
<td>• define the problem and investigate alternative approaches to a software solution</td>
<td>Teacher: • introduces concept, rationale, and methodology for project • displays and demonstrates a variety of past student projects including software demonstration, sample documentation (eg tutorials) and logbooks • defines an ‘appropriate project’, as relevant to student’s skills levels, using available software and hardware — the project should preferably include arrays, records, and some complex processing • presents categories of possible projects (eg games, business management and educational software)</td>
</tr>
<tr>
<td>H1.2 differentiates between various methods used to construct software solutions</td>
<td></td>
<td>• use a logbook to document the progress of their project</td>
<td>Teacher to review concept of logbook to document progress and idea generation. Where changes are incorporated, students should be encouraged to document these in the logbook.</td>
</tr>
<tr>
<td>H1.3 describes how the major components of a computer system store and manipulate data</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>H3.1 identifies and evaluates legal, social and ethical issues in a number of contexts</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Outcomes</td>
<td>Students learn about:</td>
<td>Students learn to:</td>
<td>Strategies, Activities and Related Resources:</td>
</tr>
<tr>
<td>----------</td>
<td>-----------------------</td>
<td>--------------------</td>
<td>-----------------------------------------------</td>
</tr>
</tbody>
</table>
| H3.2     | constructs software solutions that address legal, social and ethical issues | • select an appropriate solution | Students:  
• select two or three ideas and produce a narrative and simple storyboard for each  
• commence a logbook to include reasons for initial selection  
Teacher:  
• reviews preliminary choices and guides students in the selection of their project  
Students (with teacher guidance) further define their project in terms of:  
• programming language to be used  
• data structures  
• general algorithm approaches  
• file definition  
• a more detailed screen design for every interface in the project  
_Students should maintain logbook entries with start and completion dates and reasons for each of the above._  
Students:  
• present their ideas to the class (eg using PowerPoint, HyperStudio etc to build a prototype)  
_Other students are encouraged to give positive feedback for refinement or extension._ |
<p>| H4.1     | identifies needs to which software solutions are appropriate |                        |                                               |
| H4.2     | applies appropriate development method to solve software problems |                        |                                               |
| H4.3     | applies a modular approach to implement well structured software solutions and evaluates their effectiveness |                        |                                               |</p>
<table>
<thead>
<tr>
<th>Outcomes A student:</th>
<th>Students learn about:</th>
<th>Students learn to:</th>
<th>Strategies, Activities and Related Resources:</th>
</tr>
</thead>
</table>
| H5.1 applies project management techniques to maximise the productivity of the software development | • understanding  
  – interface design  
  – communication with others involved in the proposed system  
  – representing the system using diagrams  
  – selection of appropriate data structures | • communicate effectively with potential users at all stages of the project to ensure that it meets their requirements  
  • ensure that relevant ethical and social issues are addressed appropriately | Teacher:  
  • works with students on an individual basis to help them identify an appropriate implementation  
  • discusses with students the relevant social issues eg inclusivity, and issues related to interface design  
  
  *If necessary the teacher should guide the student to a more limited and realistic project, or the consideration of possible extensions.* |
| H5.2 creates and justifies the need for the various types of documentation required for a software solution | • applying project management techniques  
  • consideration of all social and ethical issues | | |
| H5.3 selects and applies appropriate software to facilitate the design and development of software solutions | | | |
| Students:  
  • prepare and hand in the initial system documentation (storyboard, structure chart, file structures), as well as identification of required tasks (eg file creation programs, file print programs, test data generation) | Teacher:  
  • reviews the use of a Gantt chart to:  
    – identify the separate tasks and sub-tasks  
    – document details  
    – estimate time scales and time sequencing to complete the project | | |
<table>
<thead>
<tr>
<th>Outcomes</th>
<th>Students learn about:</th>
<th>Students learn to:</th>
<th>Strategies, Activities and Related Resources:</th>
</tr>
</thead>
</table>
| A student: | • produce an initial Gantt chart | • use project management techniques to ensure that the software solution is implemented in an appropriate time frame | Students:  
• prepare and hand in an initial Gantt chart for their project, assuming five (5) months of available time  
Teachers:  
• assist students in breaking major tasks into smaller more manageable and estimable units, identifying missing necessary tasks.  

*This whole process should take no more than 4 weeks.* |
|          | • planning and design  
  – interface design  
  – selection of software environment  
  – identification of appropriate hardware | • document the software solution  
• generate a fully documented design for their project after communication with other potential users | Students:  
• break the project down into the major modules and identify the need for separate routines  
• complete a structure chart  
• identify common routines and library routines  
• set up the format for a data dictionary using Excel or a more sophisticated CASE tool to include meaningful variable name, data type, length, validation, comment and example (this will be added to over time as the need arises)  

|          | • selection of appropriate data structures  
  – production of data dictionary  
  – definition of required validation processes  
  – definition of files — record layout and creation  
  – algorithm design | • use project management techniques to ensure that the software solution is implemented in an appropriate time frame  
• use a logbook to document the progress of their project | Teachers to review work completed so far and give constructive feedback. |
<table>
<thead>
<tr>
<th>Outcomes A student:</th>
<th>Students learn about:</th>
<th>Students learn to:</th>
<th>Strategies, Activities and Related Resources:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>inclusion of standard or common routines</td>
<td>• communicate effectively with potential users at all stages of the project to ensure that it meets their requirements</td>
<td></td>
</tr>
<tr>
<td></td>
<td>use of software to document design</td>
<td>• ensure that relevant ethical and social issues are addressed appropriately</td>
<td></td>
</tr>
<tr>
<td></td>
<td>identification of appropriate test data</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>enabling and incorporating feedback from users at regular intervals</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>consideration of all social and ethical issues</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>applying project management techniques</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Systems implementation</td>
<td>Implementing the software solution by:</td>
<td></td>
<td></td>
</tr>
<tr>
<td>H6.1</td>
<td>assesses the relationship between the roles of people involved in the software development cycle</td>
<td>• implement a fully tested and documented software solution in a methodical manner</td>
<td></td>
</tr>
<tr>
<td>H6.2</td>
<td>communicates the processes involved in a software solution to an inexperienced user</td>
<td>• use project management techniques to ensure that the software solution is implemented in an appropriate time frame</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Students:</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>• start coding, using stubs as required for incomplete modules (teachers may need to assist in the selection of appropriate initial routines)</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>• access online help and user/reference manuals as required</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>• update their logbook at least weekly, to include all major milestones and difficulties, and how they were overcome</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>• accept assistance from other students and offer help to other students, as appropriate</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Outcomes</td>
<td>Students learn about:</td>
<td>Students learn to:</td>
<td>Strategies, Activities and Related Resources:</td>
</tr>
<tr>
<td>----------</td>
<td>------------------------</td>
<td>-------------------</td>
<td>-----------------------------------------------</td>
</tr>
<tr>
<td>A student:</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>H6.3 uses a collaborative approach during the software development cycle</td>
<td>– creating online help</td>
<td>• communicate effectively with potential users at all stages of the project to ensure that it meets their requirements</td>
<td>As screens are implemented, other students should be asked to offer comment in terms of design and inclusivity, including wording used on each screen.</td>
</tr>
<tr>
<td>H6.4 develops effective user interfaces, in consultation with appropriate people</td>
<td>– program testing</td>
<td>• ensure that relevant ethical and social issues are addressed appropriately.</td>
<td>Teachers:</td>
</tr>
<tr>
<td></td>
<td>– reporting on the status of the system at regular intervals</td>
<td></td>
<td>• remind students about debugging approaches including</td>
</tr>
<tr>
<td></td>
<td>– applying project management techniques</td>
<td></td>
<td>– printing variables on entry and exit from routines</td>
</tr>
<tr>
<td></td>
<td>– enabling and incorporating feedback from users at regular intervals</td>
<td></td>
<td>– referral back to design document</td>
</tr>
<tr>
<td></td>
<td>– completing all user documentation for the project</td>
<td></td>
<td>– discussion with other students (peer checking)</td>
</tr>
<tr>
<td></td>
<td>– consideration of all social and ethical issues</td>
<td></td>
<td>– formal desk checking</td>
</tr>
<tr>
<td></td>
<td>• maintenance</td>
<td></td>
<td>– use of software debugger</td>
</tr>
<tr>
<td></td>
<td>– modifying the project to ensure an improved solution</td>
<td></td>
<td>(all of the points will assist in pin pointing the source of the error, prior to correction of the code)</td>
</tr>
</tbody>
</table>
Strategies, Activities and Related Resources

End of term 1

Students:
- hand in logbook, source code, data dictionary and algorithms for assessment.

Teachers:
- assess the project to this point
- provide feedback to the students incorporating suggested improvements, methods to overcome apparent difficulties, identification of extra resources that may be accessible.

Start of term 2

Students:
- review their progress and modify their Gantt chart appropriately
- include a logbook entry to document reasons for changes to the Gantt chart, including slippages and remedial action taken (which could include limiting the project scope, allocating more practical time, seeking help from extra sources etc)
- continue coding, debugging and testing
- add logbook entries to include documentation of test data and results, and any logic changes that were required
- should remain aware of other students’ progress, so that relevant useful features may be added to their own project or assistance offered to others
- consider the relevance of online help as an integral part of specified screens
- review completed interfaces for appropriate design and social issues
- collect screen shots of completed screens.

Students need to prepare and present a formal presentation on their project (eg using PowerPoint), which should include screen shots, Gantt chart, future directions, particularly useful features of the language, major difficulties, error trapping and system documentation (Data Flow Diagram or structure chart). Other students should be encouraged to offer constructive feedback in terms of design and approaches to difficulties.
Strategies, Activities and Related Resources

Students:
- should consider all suggestions and incorporate relevant changes
- include logbook entries for any changes made as a result of the feedback, together with reasons for the changes.

Teachers:
- closely monitor the progress of individual students and their projects
- recognise students encountering difficulties or losing their way and guide them appropriately
- identify and remedy inappropriate uses of data structures, control structures, screen elements, self-documenting codes, time management etc
- recommend adjustments to the original specifications in those cases where students are struggling to cope with the load or complexity, or where students could obviously cope with a higher level of challenge within their project
- offer assistance in terms of debugging by recommending appropriate resources rather than just ‘fixing the problem’.

End of Term 2

Students:
- hand in project completed so far, with all documentation including logbook.

Start of Term 3

Teacher:
- reminds students of final project requirements including online help, tutorial, installation guide, system documentation and completed logbook
- reminds students of timeline to completion, including trial exam commitments
- provides feedback to students on project progress, including appropriate guidance as to future direction and relevant time management
- provides feedback to students on coding and debugging techniques, as well as possible areas for improvements to interface design and error trapping
- encourages interaction between students, to facilitate awareness of each other’s work and progress
<table>
<thead>
<tr>
<th><strong>Strategies, Activities and Related Resources</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Students:</strong></td>
</tr>
<tr>
<td>• redo the Gantt chart so that the project can be completed within the required time frame</td>
</tr>
<tr>
<td>• include documentation and packaging on the Gantt chart if not already there</td>
</tr>
<tr>
<td>• design the approach to be used in the creation of the tutorial (e.g., PowerPoint, HyperCard, html)</td>
</tr>
<tr>
<td>• complete implementation, including testing by themselves and other students</td>
</tr>
<tr>
<td>• complete the data dictionary to include all variables used</td>
</tr>
<tr>
<td>• tidy/complete/polish interfaces to ensure consistency</td>
</tr>
<tr>
<td>• complete all systems documentation</td>
</tr>
<tr>
<td>• produce an executable form of the software</td>
</tr>
<tr>
<td>• design and implement an icon (if relevant)</td>
</tr>
<tr>
<td>• determine requirements to run the project in an environment other than on the student’s usual computer (consider required files, directory set up and device drivers etc)</td>
</tr>
<tr>
<td>• should anticipate installation troubleshooting (e.g., missing files, insufficient memory, wrong colour settings, wrong screen size)</td>
</tr>
<tr>
<td>• create an installation guide incorporating expected desktop screen shots</td>
</tr>
<tr>
<td>• complete tutorial, utilising screen shots as appropriate.</td>
</tr>
</tbody>
</table>

**Students hand in the finished project**

Teachers should consider an appropriate completion date, around July/August, to give students adequate time for completion of the project, while at the same time allowing time for marking and submission of assessment marks to the Board of Studies.
4 Managing and Assessing a Project

4.1 Purpose of Project Management

Project management involves the coordination of a number of phases to successfully achieve identified goals. These phases include understanding the problem, making decisions, designing solutions, implementing, and testing, evaluating and maintaining.

The example project presented in this document demonstrates the phases of project management.

4.2 Tools for Project Management

4.2.1 Gantt Charts

A Gantt chart is used in the planning of a project and in monitoring its progress (refer to syllabus pp 30 and 50). It is a bar chart on which the dates and time periods for tasks and activities are scheduled. Students might find it a useful tool in their project management. (For more on Gantt charts, see Cotterell, M & Hughes, B, Software Project Management, International Thomson Computer Press, United Kingdom, 1999, ISBN 1 8503 2190 6.)

4.2.2 Logbooks

Logbooks, which could also be referred to as a journal or diary, must be used by students to detail at regular intervals, activities undertaken in the planning, design, and implementation of their projects.

The purpose of logbooks is to:

- act as a record of what a student has attempted (successfully or unsuccessfully)
- provide a formal documentation of the phases involved in project management (refer to syllabus pp 9, 35-48,)
- allow students to reflect on their progress to date by considering reasons for decisions taken.

Teachers may use the logbooks to monitor and/or assess student progress and achievement of particular outcomes.

An example of how students may approach the logbook is through the use of a scaffold. This scaffold, which identifies the main headings that students may focus their entries on, may be presented as text or in table format. The headings could include:

- date
- the task that was attempted
- how the task was achieved
- the end result of the task
- follow-up required and resulting action
- signature of teacher to confirm entry
- reasons for changes made (if appropriate).
4.3 Steps in Project Management
Refer to syllabus (pp 50-51) for the steps in designing and developing a software solution to a complex problem and to implementing it:

- defining the problem
- understanding
- planning and design
- implementation
- testing and evaluation
- maintenance.

4.4 Teaching Strategies for Group Work
Teachers need to ensure that individual and group progress is monitored in relation to the achievement of outcomes and the study of specified syllabus content.

The teacher needs to ensure that the group understands the nature of the activity, the intended outcomes and the need for individual and group goals.

This may be achieved in the following ways:
- teacher–student interviews
- teacher–group interviews
- teacher observation
- peer observation
- peer discussion.

4.5 Group Projects
4.5.1 Forming student work groups
Work groups for project work may be established in a number of ways and the method used will depend on what the purpose of the task is. Some examples of how to form groups are listed below:

- random selection — for example, by giving each student a number and having students with the same number form a group
- selection based on interests — teacher presents topics and the groups form on the basis of shared interests
- friendship groups — a commonly used method where students form their own groups. (If this method is used, the formation of heterogeneous groups should be encouraged)
- teacher-selected groups based on student skill or knowledge — in certain situations, this may ensure the achievement of specific outcomes by all groups

4.5.2 Roles of individual students
Within each group, individuals must have an identified role that has been decided on through negotiation among the members. However, depending on the purpose of the group activity, teachers may choose to allocate tasks. This might allow for the recognition of specific skills possessed by particular students, or it might facilitate the productivity of the group as a whole by altering the group dynamics.

4.6 Monitoring of Student Progress (Individual/Group)
When all students have clarified group and individuals task/s, the teacher must then develop a recording system for monitoring achievement of both group and individual outcomes.

This could be achieved using a table that indicates task descriptions, outcomes, and possible indicators of achievement. An example is shown below:

<table>
<thead>
<tr>
<th>Students</th>
<th>Syllabus Outcome: H6.3 Possible Indicators: communicates ideas and contributes to discussion of best possible solution</th>
<th>Syllabus Outcome Possible Indicators</th>
<th>Syllabus Outcome Possible Indicators</th>
</tr>
</thead>
<tbody>
<tr>
<td>Student A</td>
<td>Able to articulate ideas for solutions and suggest alternatives/improvements</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Student B</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Student C</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Student D</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

4.7 Assessment (Individual and Group)
The allocation of formal marks (as well as documentation of achievement of outcomes), should occur for tasks which are part of the school’s internal assessment program.

Teachers may provide other opportunities for students to demonstrate achievement of outcomes through normal day-to-day classroom activities. In these cases marks need not be formally allocated, but teachers should document student progress.
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:

\[
\text{LET KP = KEEPRESS} \\
\text{CALL SET RECT( RECTG,23,55,67,98)} \\
\text{LET KEYPRESS = KP}
\]

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

SLIMY SUCKERS

INITIALISE

MENU CHOICES

SAVE SCORES

PLAY

BEST SCORE

CONFIGURE

HELP

CONFIGURE

START GAME

SETUP SCREEN

DRAWSNAKE

MOVE

CHECK KEY

CHECK HIT HIMSELF

CHECK OFF SCREEN

DRAW SNAKE

CHECK HIT NO.

GROW SNAKE
### 4.9.4 Data Dictionary

<table>
<thead>
<tr>
<th>VARIABLE</th>
<th>LENGTH</th>
<th>TYPE</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>again$</td>
<td>1</td>
<td>Alpha</td>
<td>Asks the player whether he'd like to play again</td>
</tr>
<tr>
<td>back</td>
<td>numeric</td>
<td></td>
<td>variable name of the square dimensioned (back square)</td>
</tr>
<tr>
<td>backrou$</td>
<td>1</td>
<td>Alpha</td>
<td>colour of the background</td>
</tr>
<tr>
<td>backrou</td>
<td>numeric</td>
<td></td>
<td>colour of the background</td>
</tr>
<tr>
<td>colourl</td>
<td>10</td>
<td>Alpha</td>
<td>colour of the border</td>
</tr>
<tr>
<td>colour2</td>
<td>10</td>
<td>Alpha</td>
<td>colour of the snake</td>
</tr>
<tr>
<td>cover</td>
<td>numeric</td>
<td></td>
<td>used so the next number doesn't appear where the snake is located</td>
</tr>
<tr>
<td>flag</td>
<td>numeric</td>
<td></td>
<td>a flag set to know whether to increment the variable 'numb'</td>
</tr>
<tr>
<td>flag</td>
<td>numeric</td>
<td></td>
<td>program will drop out of the 'move' loop to perform a different task</td>
</tr>
<tr>
<td>helpch</td>
<td>numeric</td>
<td></td>
<td>choice from help menu</td>
</tr>
<tr>
<td>increase$</td>
<td>1</td>
<td>Alpha</td>
<td>whether the speed increases through the game</td>
</tr>
<tr>
<td>index</td>
<td>numeric</td>
<td></td>
<td>An index used in “Configuration” to be index to direction${(5})</td>
</tr>
<tr>
<td>keypress</td>
<td>numeric</td>
<td></td>
<td>the ACSII value of keypress $1$</td>
</tr>
<tr>
<td>keypress$</td>
<td>1</td>
<td>Alpha</td>
<td>the key entered by the player to make his snake move</td>
</tr>
<tr>
<td>lastkeypre</td>
<td>ss</td>
<td>numeric</td>
<td>before a new keypress is entered, the old one becomes lastkeypress</td>
</tr>
<tr>
<td>Lives</td>
<td>numeric</td>
<td></td>
<td>How many lives the player has left</td>
</tr>
<tr>
<td>mmenu</td>
<td>numeric</td>
<td></td>
<td>choice from main menu</td>
</tr>
<tr>
<td>Nam$</td>
<td>15</td>
<td>Alpha</td>
<td>name of player</td>
</tr>
<tr>
<td>Nexts</td>
<td>numeric</td>
<td></td>
<td>The level number that the player is up to</td>
</tr>
<tr>
<td>Numb</td>
<td>numeric</td>
<td></td>
<td>The counter of which is the next number the snake must 'swallow'</td>
</tr>
<tr>
<td>pause</td>
<td>numeric</td>
<td></td>
<td>flag set to say the game was just paused</td>
</tr>
<tr>
<td>rectg</td>
<td>numeric</td>
<td></td>
<td>variable name of the square dimensioned (front square)</td>
</tr>
<tr>
<td>rect</td>
<td>numeric</td>
<td></td>
<td>variable name of the square dimensioned (front square)</td>
</tr>
<tr>
<td>Rndmx</td>
<td>numeric</td>
<td></td>
<td>a random horizontal value where a number will be placed to be collected</td>
</tr>
<tr>
<td>Rndmy</td>
<td>numeric</td>
<td></td>
<td>a random vertical value where a number will be placed to be collected</td>
</tr>
<tr>
<td>Shape</td>
<td>numeric</td>
<td></td>
<td>The choice of shape by the user, It determines the shape of the snake</td>
</tr>
<tr>
<td>score!</td>
<td>numeric</td>
<td></td>
<td>the players score throughout the game</td>
</tr>
<tr>
<td>snake$</td>
<td>20</td>
<td>Alpha</td>
<td>name of player's snake</td>
</tr>
<tr>
<td>speed</td>
<td>numeric</td>
<td></td>
<td>speed at which the snake moves</td>
</tr>
<tr>
<td>pole</td>
<td>numeric</td>
<td></td>
<td>a flag set to tell if the validation of the keys was OK</td>
</tr>
<tr>
<td>upto</td>
<td>numeric</td>
<td></td>
<td>A count to see what part of the subroutine it is up to</td>
</tr>
<tr>
<td>vpos</td>
<td>numeric</td>
<td></td>
<td>the vertical position of an error message</td>
</tr>
<tr>
<td>hpos</td>
<td>numeric</td>
<td></td>
<td>the horizontal position of an error message</td>
</tr>
<tr>
<td>x</td>
<td>numeric</td>
<td></td>
<td>the current position of the snake (horizontal)</td>
</tr>
<tr>
<td>Y</td>
<td>numeric</td>
<td></td>
<td>the current position of the snake (vertical)</td>
</tr>
</tbody>
</table>

### ARRAYS

<table>
<thead>
<tr>
<th>ARRAY</th>
<th>LENGTH</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>bestname$</td>
<td>10</td>
<td>array where top ten players are sorted and then written to disk</td>
</tr>
<tr>
<td>bestscores</td>
<td>10</td>
<td>array where top ten scores are sorted and then written to disk</td>
</tr>
<tr>
<td>direction</td>
<td>5</td>
<td>array these keys are converted to ASCII values</td>
</tr>
<tr>
<td>direction$</td>
<td>5</td>
<td>array player chooses the keys he wishes to play with</td>
</tr>
<tr>
<td>done</td>
<td>10</td>
<td>array an array pointing out if a message has already been done</td>
</tr>
<tr>
<td>startx</td>
<td>640, 200</td>
<td>array The starting &quot;x&quot; position of the barrier and the number barrier</td>
</tr>
<tr>
<td>Starty</td>
<td>480, 200</td>
<td>array The starting &quot;y&quot; position of the barrier and the number barrier</td>
</tr>
<tr>
<td>endx</td>
<td>640, 200</td>
<td>array The end &quot;x&quot; position of the barrier and the number barrier</td>
</tr>
<tr>
<td>endy</td>
<td>480, 200</td>
<td>array The end &quot;y&quot; position of the barrier and the number barrier</td>
</tr>
<tr>
<td>snake$</td>
<td>640, 480</td>
<td>array Horizontal and Vertical Position of the snake</td>
</tr>
<tr>
<td>Lengt</td>
<td>9</td>
<td>array As each number is picked up, the snake grows to this certain length</td>
</tr>
<tr>
<td>Randm</td>
<td>10</td>
<td>array an array containing 10 different random messages</td>
</tr>
</tbody>
</table>
### 4.9.5 Gantt Chart

<table>
<thead>
<tr>
<th>Slimy Suckers</th>
<th>April</th>
<th>May</th>
<th>June</th>
<th>July</th>
<th>August</th>
</tr>
</thead>
<tbody>
<tr>
<td>learn to draw</td>
<td>22</td>
<td>29</td>
<td>5</td>
<td>12</td>
<td>19</td>
</tr>
<tr>
<td>get snake moving</td>
<td>16</td>
<td>10</td>
<td>17</td>
<td>24</td>
<td>31</td>
</tr>
<tr>
<td>add in numbers</td>
<td>7</td>
<td>14</td>
<td>21</td>
<td>28</td>
<td>5</td>
</tr>
<tr>
<td>grow the snake</td>
<td>12</td>
<td>19</td>
<td>26</td>
<td>2</td>
<td>9</td>
</tr>
<tr>
<td>check for numbers on</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>barriers</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>check if numbers on</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>snake</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>check if snake hits</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>itself</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>add in extra levels</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>fix all logic!</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>interface polishing</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>online help</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>documentation</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
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
            keypress$ =INKEY$
        WEND
keypress = ASC(keypress$)
IF keypress$ = "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.
flag = 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
    rndm1 = INT(RND(57)) + 1
    rndm2 = INT(RND(35)) + 4
    radmx = rndm1*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
            NEXT i
        END IF
    END IF
WEND
AND radmy >= starty(nexts,l)+10 AND radmy < endy(nexts,l) + 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

Main Menu..........................................................................................................................
High Scores.........................................................................................................................
Configuration......................................................................................................................
User Help............................................................................................................................
Exit .................................................................................................................................

Start Game

Player Details.....................................................................................................................
  Colour Options..............................................................................................................
  Speed options...............................................................................................................
  Shape Choice................................................................................................................

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.

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
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.

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 than 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](image)

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](image)

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](image)

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 (figure 2.5), and the user will have to re-enter a valid value.
Finally the game can begin. Use your configured keys to move the snake around the screen to collect numbers. Enjoy.
## 5 Preliminary Assessment Scheme

### 5.1 Example

<table>
<thead>
<tr>
<th>Assessment Components</th>
<th>Syllabus Weightings</th>
<th>Task 1 Social &amp; Ethical Issues</th>
<th>Task 2 Defining, Planning, Building and Checking Software solutions</th>
<th>Task 3 Software Development Approaches</th>
<th>Task 4 Developing Software Solutions</th>
<th>Task 5 Yearly Exam</th>
</tr>
</thead>
<tbody>
<tr>
<td>Knowledge and understanding about hardware and software, software development approaches, software development processes, social and ethical issues</td>
<td>30 5 5 20</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Design and development of software solutions</td>
<td>35 5 5 10 10</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Project management techniques, including documentation, teamwork and communication</td>
<td>15 5 5 5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Project (s)</td>
<td>20 5 15</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Marks</td>
<td>100 10 15 20 25 30</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
5.2 Task Outlines

- Task 1 – Social and Ethical Issues — Students work in pairs to research and design an appropriate work environment for a given scenario. Consideration is to be given to ergonomics, intellectual property and inclusivity. Students will present their design using presentation software.

- Task 2 – Defining, Planning, Building and Checking Software Solutions — A practical test where an original program specification is presented with a partially working code or shell. Students are to design a solution that refines the code so it satisfies the original specifications. They must desk-check, document and evaluate their solution.

- Task 3 – Software Development Approaches — Students work in groups to solve a given problem using the four different approaches (such as in Software Development Approaches in section 3.1.1).

- Task 4 – Developing Software Solutions — Major Project using pre-written modules to produce a working solution to a problem.

- Task 5 – Yearly Examination — written paper.
## 6 HSC Assessment Scheme

### 6.1 Example

<table>
<thead>
<tr>
<th>Assessment Components</th>
<th>Syllabus Weightings</th>
<th>Task 1 Development and Impact of Software Solutions</th>
<th>Task 2 Defining, Understanding, Planning and Designing</th>
<th>Task 3 Option Test</th>
<th>Task 4 Implementation, Testing, Evaluation and Maintenance</th>
<th>Task 5 Trial Exam</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Due Date Term 4</td>
<td>Due Date Term 1</td>
<td>Due Date Term 2</td>
<td>Due Date Mid Term 3</td>
<td>Due Date Term 3</td>
</tr>
<tr>
<td>Knowledge and understanding about development and impact of software solutions, the software development cycle</td>
<td>20 5</td>
<td></td>
<td></td>
<td>5</td>
<td></td>
<td>10</td>
</tr>
<tr>
<td>Design and development of software solutions</td>
<td>35</td>
<td>5</td>
<td></td>
<td>10</td>
<td>10</td>
<td>10</td>
</tr>
<tr>
<td>Project management techniques, including documentation, teamwork and communication</td>
<td>20 5</td>
<td>5</td>
<td></td>
<td>5</td>
<td></td>
<td>5</td>
</tr>
<tr>
<td>Project(s)</td>
<td>25</td>
<td>10</td>
<td></td>
<td>15</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Marks</td>
<td>100</td>
<td>10</td>
<td>20</td>
<td>15</td>
<td>30</td>
<td>25</td>
</tr>
</tbody>
</table>
6.2 Task Outlines

- Task 1 – Development and Impact of Software Solutions — A software development case study is undertaken followed by a presentation considering both social and ethical issues and current trends in software development.

- Task 2 – Defining, Understanding, Planning and Designing — Written report including all appropriate systems documentation (i.e., logbook, source code, data dictionary and algorithms) for first stage of project.

- Task 3 – Option Topic — Unit test.

- Task 4 – Implementation, Testing, Evaluation and Maintenance — Presentation and demonstration of completed project. Also to be assessed are: the program code, online help, tutorial, installation guide, system documentation and completed logbook (see Developing a Solution Package, section 3.2)

- Task 5 – Trial Examination — written paper.