

**QSITE STATE HIGH SCHOOL
YEAR 12 INFORMATION PROCESSING AND TECHNOLOGY
MINOR PROJECT – SOFTWARE ENGINEERING**

Semester 2

Assessment Conditions

- Due Date:** Friday, 3.00pm
- Method:** Preparation in class and at home. Report must be word-processed.
- Assistance:** Teacher will assist with rough drafts of proposal and complex code structures
- Assessment Criteria:** A description of the criteria for assessment of this task and the marking scheme are included on separate pages.

Task

You have the task of designing and developing a *Snakes and Ladders* computer game. The game is to be able to run from within a standard web browser and be constructed using a combination of Javascript, XHTML, CSS, and DOM. It should include the use of graphics and sound. The game should allow for at least two simultaneous players.

Required Elements

Your completed proposal must include appropriate explanation on each aspect of the software development cycle for your proposed software product as outlined below. Your **audience** for your proposal is the person who will be responsible for deciding on future funding for your project. Your **purpose** is to convince them you have a **worthwhile** concept and that your project should be supported by **further development**.

Once you have a written concept outline for the project in mind you must negotiate with your team leader (your teacher) for tentative concept approval.

A suitable application would have a simple, intuitive interface that includes a simple method for controlling the game and giving clear feedback on progress.

The proposed application should reflect the aspects of good program design that you have learnt about so far, including error trapping and confirmation. The design of the user interface is up to you, but **it is assumed that no two students would normally end up with identical concepts**. Your purpose of this major project is to demonstrate your competency within all design aspects of the software development cycle.

Note: Implementation of the design into a working application is a required aspect of this project and program code will be marked. Some assistance may be given by teachers with implementing more complex and advanced sections of the code from a prepared design. (i.e., you must know what it is you want to do before you ask how to do it in code.)

See Over for Details of Software Development Cycle...

Statement of Authorship

I confirm that that all the ideas, content and wording of this project are my own unless where otherwise clearly stated.

Student's Name: _____

Student's Signature: _____

Teacher's Signature: _____ **Submitted on:** _____

Documentation

The documentation of the solution includes the report itself plus:

- the structured design chart
- commenting in the body of the code
- instructions during the operation of the program so that the user can understand what to do
- the use of hint boxes on components and possibly in the status bar
- help file/s to be accessed from the program

1. Define the Purpose of the Project (Heading: Purpose)

Write up a clear **outline** of the **project** that you have identified for your software solution. This will include a detailed explanation of the **conditions** under which prospective users work, a discussion of the particular **needs** of the identified prospective users and the **areas** of their work that might be improved by the software solution.

2. Specify the Solution (Heading: Solution Specification)

Start with clear statement of the **objectives** for your software solution listed as bullet points.

Detail what the program is intended to do in plain English. Include:

- screen shots of your intended user interface
- a list of expected inputs/outputs
- an error-trapping table.
- a very detailed, step-by-step Plain English **explanation** of how the program will work for the user

This section should give the reader a very clear idea of what the program will look like and how it will work regardless of the programming language chosen.

3. Design the Algorithm (Heading: Algorithm Design)

Develop an algorithm design using a visual design tool (*Inspiration*) as demonstrated. Emphasis should be placed on:

Top-down design - Use a diagram based around an interface graphic or an organizational chart to demonstrate this. Clearly outline all program events to be handled, based on user interaction with the interface design.

Step-wise refinement – Use **Pseudocode** to define this. Use a separate block each module of the algorithm, i.e. event-handler or defined procedure/function. Outline all the steps required in each module of the algorithm consistent with your top-down design, making best use of indentation and appropriate naming conventions to elaborate your code structures.

4. Implement the Design (Heading: Program Code)

Implement the design in an object-oriented programming language, making sure that all aspects of the program code are laid out in a way that facilitates easy comprehension including:

- appropriate indentation
- commenting on each step in the program
- the use of meaningful names for components (eg btnQuit, txtInput)
- the use of meaningful names for variables (eg iScore, sUserName).

Include a printout of the completed program code with your project.

5. Test the Program (Heading: Testing Report)

You should arrange testing of your program by at least **3 (three)** prospective users. You should **include** a report on the outcomes of that testing including **written comments made by the trial user/s**. If bug fixes or improvements are made to the program after testing, this should be reported on also.

6. Evaluation

Write up an **evaluation** of your of your completed project in which you:

- refer directly to all of your stated objectives
- point out its strengths and where you see there might be weaknesses
- explain which parts of your planned design have been successfully implemented and what is still left to do including features that you would like to add to your project that you might not yet have learned to implement.

Knowledge					
Declarative - Knowledge of Software Engineering Structures	Recalls and uses a wide range of terms, methods, functions and concepts, demonstrating thorough understanding of software engineering practice.	Recalls and uses a substantial range of terms, methods, functions and concepts, demonstrating a good understanding of software engineering practice.	Recalls and uses a terms, methods, functions and demonstrating some understanding of software engineering practice.	Recalls and uses terms with little effectiveness demonstrating a minimal understanding of software engineering practice.	Recalls terms without being able to use them effectively.
Procedural - Knowledge of Process of Coding from Pseudocode	Consistently selects and applies knowledge, learned procedures and related concepts and principles to produce valid outcomes in a range of situations that have been previously encountered or rehearsed	Selects and applies knowledge, learned procedures and related concepts and principles to produce valid outcomes in situations that have been previously encountered or rehearsed	Applies knowledge and learned procedures to produce valid outcomes in situations which have been previously encountered or rehearsed	Applies knowledge to produce outcomes in situations that have been previously encountered or rehearsed.	Applies knowledge in situations that have been previously encountered or rehearsed
Procedural Knowledge – Language Syntax	Consistently uses functional pseudocode and programming syntax.	Mostly uses correct functional pseudocode and programming syntax.	Successfully uses correct functional pseudocode and programming syntax.	Sometimes uses functional pseudocode and programming syntax.	Uses little functional pseudocode and programming syntax.
Result	+ A -	+ B -	+ C -	+ D -	+ E -

Research and Development				
Analysis	Analyses software engineering problems very effectively	Analyses software engineering problems effectively	Analyses software engineering problems with some success	Identifies and classifies software engineering problems
Synthesis	Shows initiative in designing and developing effective, efficient and elegant solutions to unrehearsed software engineering problems	Designs and develops effective solutions to unrehearsed software engineering problems	Designs and develops functional solutions to software engineering problems	Designs, develops and evaluates limited solutions to software engineering problems
Evaluation	Evaluates software engineering problems with detailed justification, using both prescribed and self-determined criteria and standards	Evaluates software engineering problems with justification, using both prescribed and self-determined criteria and standards	Evaluates software engineering problems with some success using prescribed criteria and standards	Has difficulty making functional evaluations
Communication	Very effectively communicates ideas, solutions to and evaluations of software engineering problems clearly and cohesively,	Effectively communicates ideas, solutions to and evaluations of software engineering problems clearly.	Has some success in communicating ideas, solutions to and evaluations of software engineering problems.	Has difficulty in communicating ideas, and solutions to software engineering problems.
Result	+ A -	+ B -	+ C -	+ D -

Comment: _____