|  |  |  |
| --- | --- | --- |
|  | Faculty of Computing, Engineering and Science |  |

**Assessment Cover Sheet and Feedback Form** 2019-20

|  |  |  |  |
| --- | --- | --- | --- |
| Module Code:  CS1S466 | Module Title:  Computer Programming | | Module Team:  Mitchel Langford, Iain Shewring |
| Assessment Title and Tasks:  Set Tasks - not-time constrained 1 | | | Assessment No.  1 |
| Date Set:  **11-Nov-19** | | Submission Date:  **10-Jan-20** | Return Date:  **07-Feb-20** |

**IT IS YOUR RESPONSIBILITY TO KEEP RECORDS OF ALL WORK SUBMITTED**

|  |
| --- |
| **Marking and Assessment** |
| This assignment will be marked out of 100%  This assignment contributes to 50% of the total module marks. |
| **Learning Outcomes to be assessed** (as specified in the validated module descriptor [https://icis.southwales.ac.uk/](https://icissafe.southwales.ac.uk/studentmodules/13799/studentmodulespecifications) ):  1) Gain first hand experience of using a modern software development environment and lifecycle.  2) To apply program design and implementation techniques to satisfy a specific requirement. |
| *Provisional mark only: subject to change and / or confirmation by the Assessment Board* |

Instructions

By following the lectures, videos, and practical sessions posted on Blackboard, you will have created a basic minefield game in Visual Studio. Achieving this outcome, along with the completion of other tutorial exercises set in this term, will merit a basic pass mark.

To attain a higher mark you must engage in further independent development of the **minefield** application. Higher marks are awarded in proportion to the level of skill, innovation, robustness and functionality demonstrated in your final submission.

This is a deliberately open-ended assignment. It offers you the chance to demonstrate and highlight your individual skills, understanding, and practical capabilities in C# WinForms programming using Visual Studio IDE.

The following list provides some suggestions on possible improvements that could be made to the basic minefield game. This is not an exhaustive list, and nor is it the intention that you implement all the suggestions made here! You are strongly encouraged to develop your own independent ideas. Some of the improvements listed here are simple to achieve, while others are more challenging. You should be honest about your own abilities and try to implement those you feel comfortable with; it is better to implement simple additions well than to attempt more complex tasks that you fail to deliver.

*Suggestions for improvements to the minefield game*

Change the icons to be something better/nicer  
Add some colour border changes

Add a game reset button

Allow keypad movement control  
Add icons and meta-information to the executable

Add sounds / music effects

Add a one-off ‘quick peek’ option – shows all bombs for a second, before hiding again

Add a timer – to measure how long it took to get there (or count how many steps taken)

Add timer – to set a time limit to complete game; this could then get harder on each run

Add levels – make it harder by setting more bombs, allowing less time, etc

Add a scoring system, and/or a “Hall-of-Fame” capability

Modify the bomb-setting algorithm so that no bombs are placed in the immediate vicinity of the start/end points

Add an end of game action-replay function!

Add an option to make the trail gradually disappear as time progresses

Submission

The start of this document lists the submission date and mark return dates. Your submission should use the link posted on blackboard and found under the module’s Assessment tab.

You should submit a single Zip file. This should contain the folder inside of which your complete Visual Studio solution is contained. I must be able to extract the contents of your zip field to regenerate the folder and then be able to load and run your solution in Visual Studio. Test this is indeed the case before submission.

In addition to the Visual Studio solution files, add a Word document to the Zip file in which you succinctly describe your additions to the minefield game. This should consist at most of two sides of A4, although one side may suffice. Explain what, specifically, you have added to the basic game, and highlight anything else important about your implementation.

Marking Criteria Grid

Given the open-ended nature of this task, it is not possible to declare *X* marks are allocated to the presence of *Y* feature, as ultimately it requires academic judgement as to the sophistication and complexity of tasks attempted and degree to which they are successful. Nevertheless, the following grid provides guidance on how your work will be assessed.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Criteria** | **Fail (< 40)** | **Poor (40 – 49)** | **Reasonable (50 – 59)** | **Good (60 – 69)** | **Excellent (70 +)** |
| Code development and functionality  (weighted 50%) | Code for the basic minefield game is incomplete. Code associated with block 1 tutorials is incomplete | Code for the basic minefield game, as issued via blackboard in Week 3 is effectively complete and largely functional.  Code associated with other tutorial tasks issued in block 1 teaching are largely completed and functional | Evidence of additional code development to implement at least two of the easier suggestions for further features, or the author’s own ideas that demonstrate a similar level of technical complexity | Evidence of additional code development to implement one or more of the more complex features provided in the ‘suggestions for further features’ list. Or, coding of the author’s own ideas that demonstrate a similar level of technical complexity | Evidence of additional code development to implement the most challenging features provided in the ‘suggestions for further features’ list.  Coding of the author’s own ideas that demonstrate a similar level of technical complexity will in particularly be rewarded |
| Code presentation and style  (weighted 25%) | Poor coding style. Poor coding practices, such as default object names, non-descriptive variable names, messy and difficult to read layout. Lacks any attempt to provide comment lines | Code is readable with basic layout. Code has some attempt to include comment lines, although these may provide limited value in aiding a reader’s understanding. Some attempt to adopt meaningful object and variable names. | Widespread use of helpful comment lines. Consistent use of meaningful object and variable names. Logical code layout and ordering | In addition to all in previous category, the use if subroutines to enhance structure, readability and avoid repetitive code blocks. | In addition to requirements in previous categories further excellence shown in coding practices, clarity of layout, use of structured programming blocks, and clear and insightful internal documentation via comment lines, and so forth |
| GUI Development  (weighted 25%) | Poor layout. Use of inappropriate controls.  No attempt to enhance look and feel of user interface. No logic in interface control | Little or no development beyond that demonstrated in the supplied basic game | Some development beyond that demonstrated in the basic game supplied. Evidence of re-design of layout, adoption of alternative or additional controls appropriate to the added functionality. Attractive use of colours, fonts, etc. Basic interface logic and control to minimise potential erroneous inputs | In addition to all in previous category, clear efforts to provide an attractive, well-designed, logical, intuitive interface.  Substantial interface logic and control to minimise potential erroneous inputs | In addition to all in previous categories, further evidence of excellence in interface design and presentational quality.  Bombproof interface logic and control |

Live Demonstration

Marks awarded via the grid above may be further moderated after a live demonstration of the submitted program to the teaching staff, in which the student will be expected to highlight and discuss their added features, and to answer ensuing questions.