# Assessment Specification Software Assignment

# MOD005244

# Introduction to Game Programming

Assessment Information Forms

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| DESCRIPTIVE  TITLE | Software Assignment | | | | | Assessment  Element | 010 |
| ASSESSMENT DETAILS | You are required to design, create, test and document a console application, written in C#, to solve a problem. The specific problem will be detailed in a document published through the VLE. | | | | | | |
| Learning Outcomes | | 1 - 6 | | Weighting | 100% | Marking Approach | Fine Grade |
| WHAT IS BEING SUBMITTED | 2 copies of a USB memory stick containing:   * your program compiled as an executable (.exe) file - root * the complete project folder of your program * electronic documentation as a Word document (.doc, .docx) or Adobe (.pdf) file   attached to:  The Assignment Coversheet from e-vision (you’ll need to print this page). | | | | | | |
| SUBMISSION | ‘Due’ Date: | | 20/12/19 | Time: | 14:00 | Location | iCentre  Cambridge |
| MARKER(S): | Senir Dinar, | | | | | | |
| MODERATOR(S): | Ian Brown | | | | | | |
| FEEDBACK | Where to get your feedback: | | | Collect from … email | | ‘Post’ Date: | 20/01/2020 |

.

# Software Assignment (100%)

## Task

You are to design, implement, test and provide suitable documentation for Wry Man Air to allow bookings for the weekend ahead. There are several requirements for the software that must be met, but the choice of user interface is left to you.

The program must be written in C# CONSOLE APPLICATION and work on any University Windows PC.

. Please don’t use a Windows Forms Application for this assignment

## Scenario

***Wry Man Air*** are in need of a passenger booking application that can help manage seat bookings on their flights.

Table : Wry Man Ari Flight Times

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Flight Code | Day | Luton Departure | Edinburgh Departure | Glasgow Departure |
| WMA001 | Saturday | 05:00 | 07:15 | 08:10 |
| WMA002 | Saturday | 11:00 | 13:15 | 14:15 |
| WMA003 | Saturday | 17:05 | 19:20 | 20:30 |
| WMA101 | Sunday | 05:00 | 07:15 | 08:10 |
| WMA102 | Sunday | 11:00 | 13:15 | 14:15 |
| WMA103 | Sunday | 17:45 | 20:00 | 21:10 |

The planes only fly at the weekend, flying a triangular route from Luton to Edinburgh to Glasgow (and back to Luton). Over the course of the weekend, this route is flown 6 times (see Figure 1).

At present, the booking program for ***Wry Man Air*** only needs to allow bookings for the weekend ahead.

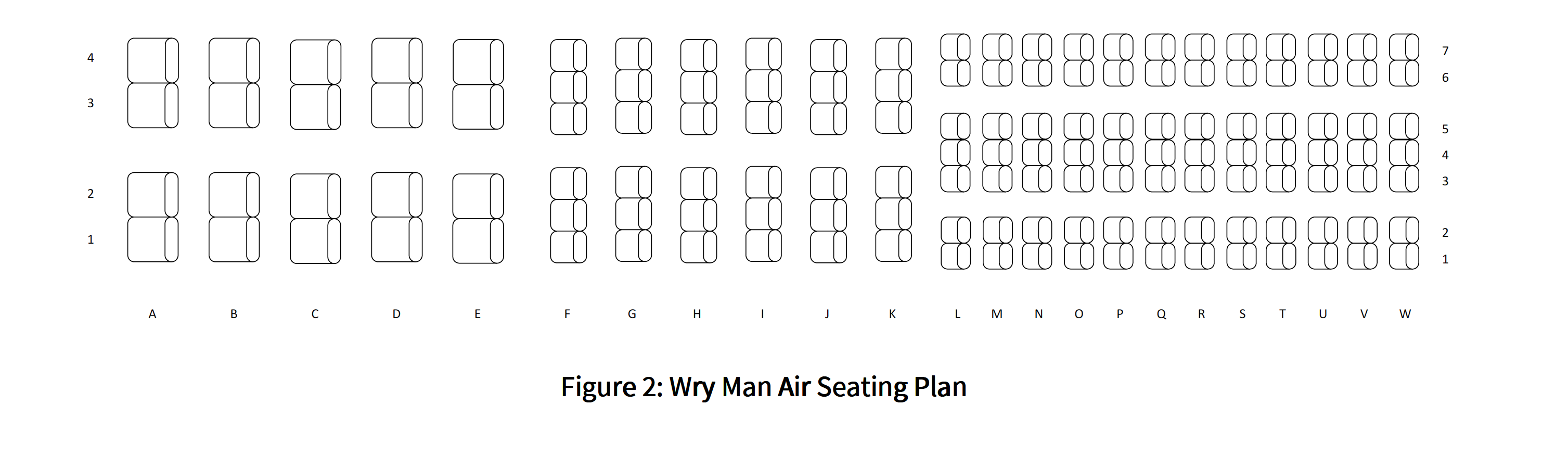
The seating layout of the plane consists of First-Class, Business Class, and Economy Class (a.k.a. Cattle Class).

Individual seats are identified by a row letter and a seat number across (1 being on the left side)

First class consists of 5 rows of two pairs of seats, with an aisle down the middle.

Business class consists of 6 rows of two triplets of seats, with an aisle down the middle.

Economy class consists of 12 rows of 7 seats, with two aisles separating the pairs of seats by the windows from three seats in the middle (see Figure 2).



When booking a party onto a flight, groups should be kept together on the same row if possible.

The following sections detail the requirements and their associated marks available.

## Program Functionality and Execution (25%)

### Basic Functionality:

The booking system must:

* Allow a passenger to choose their departure time (and therefore destination)
* Allow a passenger to choose which class they want.
* Store all booking details in a text file
* Be easy to use
* Be robust against errors and invalid inputs
* Provide clear feedback to the user

### Desired Functionality:

More marks can be earned if the booking system can:

* Refuse a booking if there is insufficient space for the passenger(s)
* Allow a passenger to book just for one or for a party
* Allow bookings for composite journeys, e.g. Luton to Glasgow. This includes allowing the same seat to be booked twice on a flight if there is no overlap
* Allow the operator to create a report as a text file (for printing) showing the Bookings for a given flight
* Automatically assign seats to a booking (Based on the rules above)

### Exceptional Functionality:

Even more marks can be earned if the booking system can:

* Read the flight schedule from a text file (see attached text file) instead of ‘hard coding’ table 1. The booking system is required to allow booking to all destinations and flight times that appear in the text file.
* Book a party of passengers in neighbouring seats as much as possible. For example, if the user is booking a party of 2, the system should reserve the seats on separate rows only if there is no other alternative.
* Conduct an in-flight Raffle ticket scheme for each flight. Each ticket that is booked is assigned a set of 6 numbers that are unique. Each of the 6 number is within in the range of 1 to 50. The uniqueness is determined across all 6 numbers (e.g. Ticket 1 is assigned 01, 02, 03, 15, 23, 45 is still considered unique when compared with Ticket 2: 01, 02, 03, 15, 23 and 52). Once a given flight has sold more than 90% of the available tickets (simulating the plane taking off) the 6 winning number are randomly generated by the programme and the winning tickets (Seat number) are generated. First prize (All 6 numbers), second prize (5 numbers), third prize (4 numbers). If more than one ticket wins a given prize, all the winning tickets (seat number) should show.

## Program Architecture and Authorship (25%)

### The code should:

* Be Object Oriented in its design and include a minimum of 5 classes.
* Be easily maintainable, and extendable to more frequent flight times.
* Clearly separate UI functionality objects from the data manipulation objects
* Name classes, variables, methods etc. with meaningful, clear names using consistent, appropriate capitalisation
* Be well documented with both normal // comments within functions and detailed /// comments on methods, classes and other definitions
* Have a consistent, appropriate layout including good use of indentation, white-space and individual files for each class

## Submission & Presentation (10%)

The work must be submitted according to the following instructions. This ensures reduced risk for you and maximum convenience for the marking staff.

Print a copy of the cover sheet and receipts from e-vision. Do not print anything else (code, documentation etc).

#### Create two copies of a USB Stick containing

* The booking program compiled as an executable (.exe) file. Placed in the root of the memory stick
* The complete project folder of the source code for the program
* The documentation in electronic form as a Word document (.doc,
* .docx) or Adobe (.pdf) file if written, or MP4 file if a video (Links to YouTube or other hosting services are not acceptable). Excel workbooks (.xls, .xlsx) are acceptable for the testing section.
* Check the USBs work: it is your responsibility to ensure that the disks you submit are readable and that the files on them can be copied and opened. Staff will attempt to use the first USB and the second USB on two University- owned computers. If after this point the USBs still cannot be read, a mark of zero (0) will be awarded.
* Hand this in to the iCentre (Cambridge Campus) before the deadline. Keep a copy of your work and the receipt in a safe place until the end of the academic year.
* **Clearly Label your USB sticks with your Student ID Number (and if there is space the module code as welll** (005244))
* **.**

## User Guide (10%)

A ‘getting started’ User Guide for the program.

This can be **written** (in a Word doc) or in the form of a video tutorial (maximum 2 minutes length). Clarity, structure and presentation will be important. The length should be no less than 150 words and no more than 400.

## Narrative of design (15%)

The design can be explained through a video (maximum 5 minutes length) or a prose description (maximum 2 sides, which may include a DFD). The code is expected to match this design.

You may incorporate pseudocode or other design methods that you are familiar with.

The design document should be detailed enough that another developer could replicate your solution in any programming language.

## Evidence of testing (15%)

Full documentation of the tests carried out throughout the development of the software.

A proposed test plan layout can be seen in the module notes. You need not copy this exactly, but whatever plan you choose to use, it must be easily read, accurate, and thorough i.e. all operations in your code are expected to have been exhaustively tested and recorded with enough detail to repeat the tests.

You may choose to compile this documentation using an Excel spreadsheet if you prefer.

Other Notes

The intended audience for your documentation is to be considered that of your fellow students.

You should include references to any sources you draw upon.

A program that does not perform all the requirements will not necessarily be considered a failure – it can still earn marks for what it can do and how it does it. Clearly document any shortcomings.

This assignment is ‘**equivalent to’** a 5000-words essay.

The deadline for submission is 2pm, 20th December 2019.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Mark Scheme | 70+ Excellent – Grade 60 (+) | Good – Grade 50 (+) | Adequate – Grade 40 (+) | Insufficient–Grade 0 – 40 |
| Program Functionality and Execution 25% | All basic functionality and most desired functionality implemented and working.  Simple and intuitive to use.  Clear meaningful instructions and error messages.  Robust and reliable, with all exceptions handled without crashing. | All basic functionality working, some additional functionality working.  Usable program.  Majority of errors handled without crashing.  Clear messages and instructions to the user. | Most basic functionality working.  User experience complex or inconsistent. Messages/instructions unclear or not enough of them.  Frequent exceptions (crashes) and/or unhelpful error messages. | Majority of required functionality not working. Confusing user experience |
| Program Architecture and Authorship 25% | Excellent adherence to coding standards i.e. indentation, use of comments, naming and capitalisation.  Code easily maintainable and easy to follow. Appropriate use of ‘new’, ‘untaught’ features.  Clear and sensible use of classes and methods. Good separation of UI and data functionality and strong use of OO principles of encapsulation, inheritance and polymorphism. | Code easy to follow. Maintainable and well presented.  Sound application of taught material.  Use of classes and methods but little evidence of OO principles.  Weak separation of functionality. | Code confused and difficult to follow intended behaviour.  Limited adherence to good coding practices or code which is otherwise difficult to maintain.  Attempt at implementing taught material with limited success.  Limited or confusing use of classes and methods. | Little or no attempt at solving the problem or most code submitted is auto-generated.  Lacking in basic topics as covered in module.  Classes poorly used if at all. Methods used sparingly or not at all. |
| Submission Presentation  10% | All submission requirements followed. Self- Assessment thoughtfully completed. | Majority of submission requirements followed. Self- assessment and checklist completed. | Self-Assessment or checklist only partly completed. Disks difficult to safely extract. Excess printed material or unnecessary binding/wallet. Used USB not disk. | Little or no adherence to submission requirements. No self assessment form, missing backup or loose CD. |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| User Guide 10% | Clear and easily read/watched. Effectively instructs the user on the use of the program.  Extra value such as troubleshooting, future development plans, screenshots. | Effectively instructs the user on the use of the program.  Sound structure, good presentation.  Some added value. | Covers the main features of using the program.  Structure is hard to follow or not logical, or presentation is haphazard. | Poorly set out or inadequate guide to even the basic use of the program. |
| Evidence of design  15% | Clear and easy to follow design. Could be used for majority of languages.  All functionality accounted for, potentially with extra quality of life features.  Layout and presentation easy to follow. Excellent presentation, grammar and structure. References and citations included. | ‘Usable’ design that will produce ‘similar’ working application in same language.  All required and some further functionality handled by the design.  Good presentation, grammar, structure and references. | Design not reflected in the program created, or required functionality left out.  Poor presentation. | Design document repeats assignment brief, or does not explain how the problem has been approached.  Presentation and structure confused. |
| Testing 15% | Significant number of tests recorded in detail. Test data and procedure clearly evidenced.  Good evidence of boundary testing and repeated testing where prior tests have failed. Evidence of testing throughout development process.  Clear layout of information, with additional analysis | Good number of meaningful tests recorded with few details missing, or smaller number of tests at exceptional detail. Evidence of failed tests. Evidence of range of inputs being used.  Clear, structured layout of information. | Reasonable number of meaningful tests recorded but with many details missing (e.g. test data, steps taken, date & result information). Small number of tests at a higher level of detail with few details missing.  Functional layout of information. | Little or no evidence of testing, or trivial/similar tests repeated.  Confused layout of results. |