MANCHESTER METROPOLITAN UNIVERSITY
School of Computing, Mathematics & Digital Technology
ASSIGNMENT COVER SHEET

Manchester Metropolitan University

Unit:	6G4Z1011 Programming
Assignment set by:	Dr David McLean
Verified by:	Nick Whittaker
Moderated by:	Nick Whittaker
Assignment number:	1CWK50
Assignment title:	Shoot 'em Up
Type: INDIVIDUAL	INDIVIDUAL – see plagiarism statement
Hand-in format and mechanism:	via Unit area on Moodle
Deadline:	As indicated on Moodle.

Learning Outcomes Assessed:

- apply the main structuring features of the chosen high level programming language(s) to solve a variety of problems
- design well-structured solutions to a problems of varying complexity using appropriate methods.
- implement well-structured solutions to a variety of problems using the appropriate techniques and a high-level programming language
- apply object oriented design to model a variety of real world (type) problems
- construct and apply suitable software tests

It is your responsibility to ensure that your work is complete and available for assessment by the date given on Moodle. If submitting via Moodle, you are advised to check your work after upload; and that all content is accessible. Do not alter after the deadline. You should make at least one full backup copy of your work.

Penalties for late hand-in: see Regulations for Undergraduate Programmes of Study: http://www.mmu.ac.uk/academic/casqe/regulations/assessment.php. The timeliness of submissions is strictly monitored and enforced.

Exceptional Factors affecting your performance: see Regulations for Undergraduate Programmes of Study : http://www.mmu.ac.uk/academic/casqe/regulations/assessment/docs/ug-regs.pdf

Plagiarism: Plagiarism is the unacknowledged representation of another person's work, or use of their ideas, as one's own. MMU takes care to detect plagiarism, employs plagiarism detection software, and imposes severe penalties, as outlined in the Student Handbook (http://www.mmu.ac.uk/academic/casqe/regulations/docs/policies_regulations.pdf and Regulations for Undergraduate Programmes (http://www.mmu.ac.uk/academic/casqe/regulations/assessment.php). Bad referencing or submitting the wrong assignment may still be treated as plagiarism. If in doubt, seek advice from your tutor.

Assessment Criteria:	Indicated in the attached assignment specification.
Formative Feedback:	Checkpoint 1: submission of portfolio wk9 Defender Checkpoint 2: week 18 formative deadline
Summative Feedback format:	Example marking grid provided in attachment
Weighting:	This Assignment is weighted at 50% of the total unit assessment.

Please read ALL the specification before starting and before hand-ins
Learning Outcomes Assessed:

- apply the main structuring features of the chosen high level programming language(s) to solve a variety of problems
- design well-structured solutions to a problems of varying complexity using appropriate methods.
- implement well-structured solutions to a variety of problems using the appropriate techniques and a high-level programming language
- apply object oriented design to model a variety of real world (type) problems
- construct and apply suitable software tests

Sections

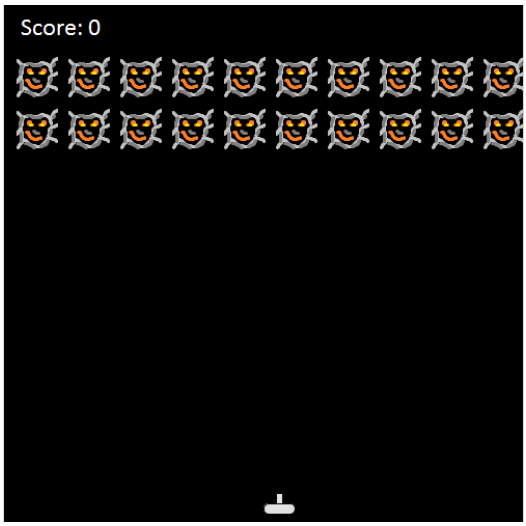
1. Employability
2. Formative and Summative Submission
3. Application (the specification)
4. Marking Scheme
5. Hand-in (procedure)
6. Notes and Help
7. Plagiarism
8. Example summative marking grid/scheme

1. Employability statement:

This assignment will help develop your problem solving and programming skills on a relatively large and complex application involving multiple Classes. All the techniques used are industrially relevant and a completed application can be used as evidence of your abilities and skills for your CV and future placement and career interviews. It would be wise to produce a digital artefact (e.g. screen recording) which can be referenced in your CV or covering letter. Your finished application could include some test code that calls and tests the methods within a class for robustness (however these can be isolated from the main game loop or commented out in the final submission).

2. Formative and Summative Submission

You will be provided with **verbal feedback** on your Defenderz portfolio exercise which is the first stage of this assignment. You must hand-in your work to date on the **formative submission date**. Your work will be checked at this point and you will be provided with **written feedback** to enable you to improve your submission. You can then resubmit your improved submission on the final hand-in date, but your work must be accompanied with a record of the changes you have made in light of the feedback you were given. This record should consist of a written description and highlighted sections within your code (e.g. by comments). You can also ask for formative **verbal feedback** during the assignment based labs (2nd term) and in support sessions. You can also seek staff support from your lab tutors (see staff availability by staff doors) and in support labs (advertised on moodle).



3. Application

You must design and Implement in **Processing (Java)**, a Space Invaderz type game, with a user controlled defender and computer controlled invaders. You can select any theme you like for the game (e.g. a gardener shooting greenfly) but your INVADERS, MUST BE YOUR OWN INDIVIDUAL DESIGN, the game cannot be simply be a copy of the original.

Defender must be able to

- Move left to right (could also move up and down)
- Shoot a missile up the screen

Invaders must be able to

- Move around the screen in some set pattern
- Cause Defender to die by reaching a set point (e.g. bottom of the screen)

Invaders could be able to (must for >70)

- Drop missiles
- Be shot (by the defender) and disappear from the screen

Solution must include:

- Classes to handle the defender, the Invaders, missile(s). You may add other Classes where suitable.
- Classes should all contain a variety of appropriate methods including at least one constructor.
- At least one ArrayList or (preferably) a 2D array (must for >60) to handle multiple objects of the same class.
- At least one class should contain a function method

Optional additions (will increase you mark within a grade range)

- Splash screen at start
- Game levels
- Player lives
- Scoring – current score, best score
- Complex movement patterns
- Intelligent invaders

4. **Marking Scheme : All features listed must be present to achieve each range of marks**

To pass (40-50) a simple working game (you should comment out code that causes errors)
At least one invader that can move, defender can shoot, defender can move (e.g. key presses).
50-60 all the above and but multiple Invaders, game can end
60-70 all the above and 2 Dimensional array of Invaders, multiple Invader images displayed in sequence (an animation sequence), Invaders can be shot
70-80 all the above and Invaders can drop missiles, Defender can be shot
80+ all the above and a fully working game, multiple levels, scoring, optional additions

The following criteria will be considered when marking your work:

- OO design - Complete and Working Classes, appropriate use of private and public accessibility .
- Reusability of classes (and methods), minimal global variables
- Refactored code – well-structured code, intuitive variable names,
- Presentation: Consistent indentation of code blocks, source code contains informative comments.

5. **Hand-in** :

Formative: Submit to moodle a zip file containing your solution directory, and all the associated files (code file(s) which will run without errors, and any image files ect). The zip file should consist of your name and student number followed by the grade you believe you have achieved e.g. DavidMcLean99700733_75.zip where my code has ALL the features indicated in the 70-80% range.
Summative: Submit to moodle a zip file containing your solution directory, and all the associated files including:

- a list of changes (text document describing any changes since the formative submission)
- code file(s) which will run without errors,
- any image files, etc.

The zip file should consist of your name and student number e.g. DavidMcLean99700733.zip

Please note that the submission inbox on Moodle will not accept submissions larger than 20MB. Your zip file is unlikely to be this large unless you have used very large image files. Please check in good time that your work will fit within the size limit.

6. Notes and help

Re-read your lecture notes – we've covered similar ideas within the lectures and labs (specifically week 9).
Use top down design (as taught throughout the first term lectures) to help you implement your various methods.
Class design : carefully consider all the members within your Class(es) do they have sensible names (obvious what they are intended to do) are they all necessary, would more members simplify your code?
Does each method perform only one task? is all the information it needs passed as parameters?

7. Plagiarism

There are a number of online implementations of this and similar games. While it is OK to review these for ideas, it is NOT OK to copy whole or parts of these code examples and pass them off as your own. Similarly it is NOT OK to borrow code in whole or part from your peers, both of you would be guilty of plagiarism. We will use an AUTOMATED PLAGIARISM CHECKER to compare your submissions against other students work and online examples.

8. Example Summative marking scheme:

Features must all be present to achieve the range of marks				**Base Mark**
Working game	1 Invader : moves	defender shoot	defender move	40-50%
	Multiple Invaders		game can end	50-60%
Invader(sequence images)	Invaders shot, disappear	2D Array Invaders		60-70%
Defender - shot	Invader missiles			70-80%
scoring	Player lives	Splash screen	Multiple levels	80-90%
explosions	Intelligent Aliens	Complex Invader movements	Others	90-100%
The following Coding concepts are deemed as good programming practice, those highlighted were observed in you code and have brought your mark up or down within the range				
Multiple Classes	Constructor(s)	methods	Public/private	Inheritance
	Meaningful Variable names	constants		

Well structured Indentation Intuitive order	Well factored – procedures/parameters No duplication Easily read	Comments where necessary	Enum set – switch case	

Mark:

Strength(s):

Weakness(es):