Manchester Metropolitan University

Department of Computing and Mathematics

ASSIGNMENT COVER SHEET

Unit code & title:	6G4Z0020 Programming
Assignment set by:	Dr David McLean
Assignment ID:	2 Cwk 50%
Assignment title:	Mouse Click Game
Type: (Group/Individual)	**Individual**
Hand-in deadline:	assessment week (ending Fri 13th Jan, please see Moodle – assessment area for your personal deadline).
Hand-in format and mechanism:	Electronic submission (zip file of code) via Moodle

Learning outcomes being assessed:

- Apply computational thinking and fundamental programming concepts to solve problems
- Design and implement well-structured solutions to problems of varying complexity using appropriate methods, including Object Oriented techniques.
- Adopt a reasoned approach to identify and rectify software defects in simple programs.

Note: it is your responsibility to make sure that your work is complete and available for marking by the deadline. Make sure that you have followed the submission instructions carefully, and your work is submitted in the correct format, using the correct hand-in mechanism (e.g. Moodle upload). If submitting via Moodle, you are advised to check your work after upload, to make sure it has uploaded properly. Do not alter your work after the deadline. You should make at least one full backup copy of your work.

Penalties for late hand-in: see Assessment Regulations for Undergraduate/Postgraduate Programmes of Study on the Student Life web pages. The timeliness of submissions is strictly monitored and enforced. All coursework has a late submission window of 7 days (i.e. 5 working days), but any work submitted within the late window will be capped at 40%, unless you have an agreed extension. Work submitted after the 7-day late window will be capped at zero, unless you have an agreed extension. See below for further information on extensions.
Please note that individual tutors are unable to grant extensions to coursework.

Extensions: For most coursework assessments, you can request a 7-day extension through Moodle. This will not apply to in-class tests, presentations, interviews, etc, that take place at a specific time. If you need

a longer extension you can apply for an Evidenced Extension. For an Evidenced Extension you **MUST** submit 3rd party evidence of the condition or situation which has negatively impacted on your ability to submit or perform in an assessment.

Plagiarism: Plagiarism is the unacknowledged representation of another person's work, or use of their ideas, as one's own. Manchester Metropolitan University 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 (https://www.mmu.ac.uk/academic/casqe/regulations/assessment/docs/ug-regs.pdf). Bad referencing or submitting the wrong assignment may still be treated as plagiarism. If in doubt, seek advice from your tutor.

If you are unable to upload your work to Moodle: If you have problems submitting your work through Moodle you can email it to the Assessment Team's Contingency Submission Inbox using the email address submit@mmu.ac.uk. You should say in your email which unit the work is for, and ideally provide the name of the Unit Leader. The Assessment team will then forward your work to the appropriate person. If you use this submission method, your work must be emailed by the published deadline, or it will be logged as a late submission. Alternatively, you can save your work to your university OneDrive and submit a Word document to Moodle which includes a link to the folder. It is your responsibility to make sure you share the OneDrive folder with the Unit Leader.

As part of a plagiarism check or in the case of an exceptional submission, you may be asked to attend a meeting with the a member of the unit delivery team, to discuss (and provide verbal feedback) or to explain your work (e.g. explain the code in a programming assignment). If you are called to one of these meetings, it is very important that you attend.

Assessment Criteria:	Indicated in the attached assignment specification.
Formative Feedback:	You can seek formative **verbal feedback** by asking your lab tutor within your weekly lab session. The week 11 lab session will also be partly set aside for you to get formative feedback on you game so far.
Summative Feedback Format:	Summative feedback will be provided through moodle, with individual and group feedback.

Please read ALL the specification before starting and before hand-in

Sections

1. Employability Statement
2. Game Application (the specification)
3. Hand-in (procedure)
4. Plagiarism
5. Example summative marking grid/scheme
6. Hints & help, Getting Started

1. Employability Statement

This assignment will help develop your problem solving and programming skills on a relatively large and complex application involving ***multiple classes*** and objects. 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 or web enabled executable) which can be referenced from your CV or covering letter.

2. Game Application

You must design and Implement in **Processing (Java)**, a 2-D game, with a goal (somewhere on the screen), a group of computer controlled animated enemy objects, which move towards the goal and a mouse-controlled object that takes some action against the enemies. You can choose any game scenario that fits this description (if unsure ask your lab tutor). Your code MUST contain a minimum of 2 classes, though more classes should be added as necessary.
Some possible example game concepts –

- "Fly Swat" Goal is a cake (middle of screen), animated enemy objects are flies which appear randomly around the screen and converge on the cake. User swats flies via a mouse controlled swat and clicking the mouse on the flies.
- "Scare Crow" Goal is corn field at bottom of screen, enemy objects are birds converging on the corn, clicking on a bird causes the word **Bang** to appear on the screen, scaring the bird away.
- "Troll Attack" Goal is a castle on right of screen, enemy objects are trolls which walk towards the castle. A knight moves wherever the mouse is clicked (runs there) and on colliding with a troll removes it from the screen.
- "Ice Hockey" enemy objects are pucks, a keeper moves to wherever the mouse is clicked (runs there) and collides (stops the pucks) hitting the goal.

In all examples the game would end when a specific number of enemy objects reach the goal.
All the techniques you need to solve this will be covered in the lecture notes and lab exercises.

There are no marks available for the quality of the graphics used, just for the animation sequences and coding techniques, so leave any images (gif jpg etc) until later in your application development.

Optional additions

- Game levels
- Player lives
- Scoring
- Multiple animated sequences (must differ in sequencing) – e.g. spinning, explosion, walk left, walk right etc.

3. Hand-in

Submit to Moodle a **ZIP** file containing your solution **directory**, and all the associated files including:

- code file(s) which will run without errors – code with errors should be commented out

- 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 100MB. Your zip file is unlikely to be this large unless you have used very large image files. Please check in good time that your zip-compressed work will fit within the size limit.

4. Plagiarism

This assessment must be a product of your own efforts. We will use a **PLAGIARISM CHECKER** to compare your submissions against other student submissions and any similar online examples. Last year some students were found to have high degrees of similarity in their submissions and had to undergo formal plagiarism hearings in some cases resulting in expulsion (see Student Handbook). Do not copy blocks of code or allow blocks of code from your own work to be seen or copied by others.

5. Summative Marking Scheme:

All features listed must be present to achieve each range of marks. For example, to get over 70%, all features from the previous bands (40-50),(50-60), (60-70) must be in your code, including some features in the 70-80 band. Overall quality of your code will also affect your grade, bringing the mark up or down (specifically see the Additions and Code Quality feedback table for the concepts we will be looking for).

Base Mark	**Features Required** (starting at top, tick off each criteria working down the list)
40%	**All** of the following to Pass (40%): ☐ Goal drawn on screen ☐ Minimum 2 classes ☐ At least one enemy object that moves towards the goal ☐ Simple working game (comment out code that causes errors) in Processing ☐ Something clearly happens when the mouse is clicked (affects an enemy object or causes a player object to move which affects enemy objects on collision)
40-50%	**All** of the above, and some of the following: ☐ At least 3 enemy objects (on screen) that move towards the goal ☐ Enemy objects freeze or disappear when hit by a mouse click ☐ Working Collision *function* method(s) : for the goal and for the mouse_click
50-60%	**All** of the above, and some of the following: ☐ Splash or game over screen (draw does different things at different times) ☐ An ArrayList (or array) of enemy objects ☐ Animated sequence of images for the enemy objects (appears to walk, spin, fly, etc)
60-70%	**All** of the above, and some of the following: ☐ A 2nd type (class) of enemy objects that must be avoided by the player (collision involves loss of life or game end) ☐ Class-inheritance for different attacker types (perhaps other classes) ☐ File handling – high score(s) saved and read from file
70-80%	**All** of the above, and some of the following ☐ Array of PImages for animation sequence(s) ☐ Exhibits some polymorphism with the array/arrayList of enemies
80%+	**All** of the above, and some of the following: ☐ collision animation sequence(s) (e.g. explosion)

	☐ Complex enemy movements (collaboration to make the game more difficult, enemies move in different patterns) ☐ Polymorphism for most (or all) game entities ☐ Use of an Interface or abstract class ☐ Refactored, maintainable code

Additions and Code Quality

The following table will be completed during marking for feedback. We will Highlight the concepts or features found in your submission. The inclusion or lack of these may increase or decrease your overall grade.				
Onscreen: Score lives	Game levels	Animated sequence of images: • single • Multiple	Classes: Well structured Members, Suitable methods, Appropriate number	Constructor(s): Single Multiple
Meaningful Variable names: • Mostly • All	Comments where necessary Largely self-documented code	constants - appropriate use	Switch case – game mode	Function method(s), appropriate use
Well factored – procedures/parameters • Names • No duplication • Easily read	Well-structured code: • Mostly • All Indentation Intuitive order	Concise efficient code	Enum set e.g. game modes	Public/private

6. Hints & Help, Getting Started

In week 6 we had the Defenderz lab exercise. Start by attempting the Defenderz lab exercise (read accompanying teaching material "Developing Multiple Classes and Objects") here you had 2 or 3 classes that can be modified to get started with this assignment. Remember the weeks prior to this with teaching material entitled "Using a Class" & "Writing a Class."

Start simple, add one thing at a time and test it – check it works properly.

Remember that when a collision occurs (e.g. when the mouse click occurs on an enemy object) then we need to get rid of the object (set the instance equal to **null**). This must be handled from code OUTSIDE the class (from the draw event or a procedure called from the draw event). A collision function method should be called that can RETURN a value to the calling code, so the calling code knows to remove it (see notes on classes with arrays).

There are many sources of images (Paint 3D) can be used to produce your own or edit existing images. There are free online images for background and animated image sequences (https://opengameart.org/), these may need to be credited.

Re-read your lecture notes – we've covered similar ideas within the lectures and labs (specifically the material on Multiple Classes).

Use top-down design 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?