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:	
Moderated by:	Dr - Alan Crispin
Assignment number:	1CWK40
Assignment title:	Mouse click game
Type: INDIVIDUAL	INDIVIDUAL – see plagiarism statement
Hand-in format and mechanism:	via Unit area on Moodle (near top of page – under Unit Assessment)
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

---

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 40% of the total unit assessment.

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

Sections

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

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. Game Application

You must design and Implement in **Processing (Java)**, a 2-D game, with a mouse-controller a goal (somewhere on the screen) and a group of computer controlled animated enemy objects, which move towards the goal. You can choose any game scenario that fits this description (if unsure ask your lab tutor).

Some possible examples –

- "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 by clicking the mouse on them.
- "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 have already been (or will be) covered in the lecture notes and lab exercises.

player must be able to

- Click on the screen and remove an enemy object at that position, or cause a defender (player controlled) object to move to that position which collides with enemy objects and causes them to be removed from the game.

Enemy objects

- Must be able to move around the screen (in some set pattern) towards a goal (in the simplest case the goal could be one side of the screen).
- Should have an animated sequence, e.g. appear to walk, swim, fly etc. towards the goal
- Should cause the game to end (or loss of player life) by multiple objects reaching the goal

Solution must include:

- At least two classes, one to handle the enemy object and one to handle the goal. You may add further Classes where appropriate.
- Classes should all contain a variety of suitable methods including at least one constructor.
- At least one ArrayList/array to handle multiple objects of the same class.
- At least one class should contain a **function** method

Optional additions (will increase your grade range)

- Splash screen at start
- Game levels
- Player lives
- Scoring – current score, best score
- Complex movement patterns (avoiding the mouse position)
- Multiple animated sequences – e.g. walk, explosion, attack the goal etc
- Intelligent enemy objects

4. **Marking Scheme : All features listed must be present to achieve each range of marks** (i.e to get 50+, there must be more than one enemy on the screen and the game ends when something specific happens)

To pass (40-50) a simple working game (you should comment out code that causes errors)
A goal drawn on the screen, at least one enemy that can move towards the goal, something clearly happens when an enemy is clicked.
50-60 all the above and enemy objects disappear (or freeze) when hit by a mouse click (or collision with player character), multiple enemy objects on the screen simultaneously, game ends when a specified number of enemy object reach the goal).
60-70 all the above and, an animated sequence of images for the enemy objects (e.g. walking), an arrayList or array of enemy objects, game over message screen when game ends, small goal (i.e. not a whole screen edge).
70-80 all the above and scoring, a splash screen, a further sequence of animated images (for example could be an explosion when hitting an enemy object or an end of game sequence when enemies have got to the goal), player lives.
80+ all the above and a player character that moves to the mouse clicked position, enemy objects disappear when colliding with the player character, multiple levels, class inheritance. Well-structured code

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 etc). 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 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.

6. Plagiarism

We will use an AUTOMATED PLAGIARISM CHECKER to compare your submissions against all other student submissions and any similar online examples. This system is impervious to changes of variable/procedure names, moving blocks of code around etc. Last year over 20 students were found to have high degrees of similarity in their submissions and had to undergo formal plagiarism hearings and were dealt with harshly (see Student Handbook). Do not copy large blocks of code or allow blocks of code from your own work to be copied by others.

7. Example Summative marking scheme:

Features must all be present to achieve the upper mark in a range (i.e to get 60, there must be multiple enemies on the screen which are removed from the game when clicked, the game can end.				**Base Mark**
Working game	Goal on screen	1 enemy moves to goal	Enemy clickable	40-50%
Multiple Enemies	Enemies disappear/freeze	game end		50-60%
Enemy animated (sequence of images, e.g. walking)	ArrayList/Array enemies	Game over message	Goal must be smaller than screen (width/height)	60-70%
scoring	Splash screen	2nd animated sequence (e.g. explosion)	Player lives	70-80%
Multiple levels	Player character with collision detection	inheritance		80-100%

The following Coding concepts are deemed as good programming practice, those highlighted were observed in you code, those in red were expected but missing. These have brought your mark up or down within the range				
Multiple Classes	Constructor(s)	methods	Public/private	
Meaningful Variable names	constants	Use of Functions	Switch case – game mode	polymorphism
Well structured Indentation Intuitive order	Well factored – procedures/parameters No duplication Easily read	Comments where necessary	Enum set – game modes	

Mark:

Strength(s):

Weakness(es):

8. Hints & Help, Getting Started

Start by attempting the Defenderz portfolio exercise (week 9 – read accompanying lecture notes "Developing Multiple Classes and Objects") here you had 2 or 3 classes that can be modified to get started with this assignment.. Remember the two weeks prior to week 9, lead in to this with lectures entitled "Using a Class" & "Developing a Class." Mouse click events were dealt with in week 4 Animation.

Add one thing at a time and test it – check it works properly.

Remember that when a collision occurs (e.g. when an enemy is clicked) then we need to get rid of the enemy (set the instance equal to null). This must be handled from code OUTSIDE the class (from the draw event), so the object needs to RETURN a value to the calling code, so the calling code knows to remove it (see lecture week 9):

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?