**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:** | Product Assembly Game |
| **Type: (Group/Individual)** | **Individual** |
| **Hand-in deadline:** | assessment week (ending Fri 12th 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**:

**Penalties for late submission**

The timeliness of submissions is strictly monitored and enforced.

All coursework has a late submission window of 7 calendar 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 ‘Assessment Mitigation’ below for further information on extensions.

**Please note that individual tutors are unable to grant extensions to assessments.**

**Assessment Mitigation**

If there is a valid reason why you are unable to submit your assessment by the deadline you may

apply for assessment mitigation. There are two types of mitigation you can apply for via the unit area on

Moodle (in the ‘Assessments’ block on the right-hand side of the page):

• **Self-certification**: does **not** require you to submit evidence. It allows you to add a short extension to a deadline. This is not available for event-based assessments such as in-class tests, presentations, interviews, etc. You can apply for this extension during the assessment weeks, and the request must be made **before** the submission deadline.

* **Evidenced extensions:** requires you to provide independent evidence of a situation which has impacted you. Allows you to apply for a longer extension and is available for event-based assessment such as in-class test, presentations, interviews, etc. For event-based assessments, the normal outcome is that the assessment will be deferred to the Summer resit period.

Further information about Assessment Mitigation is available on the dedicated Assessments page: <https://www.mmu.ac.uk/student-life/course/assessments#ai-69991-0>

**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 Code of Conduct](https://www.mmu.ac.uk/student-case-management/guidance-for-students/student-code-of-conduct/) and [Regulations for Undergraduate Programmes](https://www.mmu.ac.uk/student-life/course/assessments#ai-63930-1). Poor referencing or submitting the wrong assignment may still be treated as plagiarism. If in doubt, seek advice from your tutor.

**As part of a plagiarism check, you may be asked to attend a meeting with the Unit Leader, or another member of the unit delivery team, where you will be asked 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.**

**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](mailto:submit@mmu.ac.uk). You should say in your email which unit the work is for, and 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 **before the published deadline**, or it will be logged as a late submission.Alternatively, you can save your work into a single zip folder then upload the zip folder 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, or it will not be possible to mark your work.**

**Assessment Regulations**

For further information see [Assessment Regulations for Undergraduate/Postgraduate Programmes of Study](https://www.mmu.ac.uk/student-life/course/assessments#ai-63930-2) on the [Student Life web pages](https://www.mmu.ac.uk/news-and-events/news/story/?id=16239).

|  |  |
| --- | --- |
| **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 and Report (the specification)
3. Hand-in (procedure)
4. Example summative marking grid/scheme
5. Support, Hints & help, Getting Started

# 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.

1. **Game Application and Implementation Report**

You must design and Implement in **Processing (Java)**, a 2-D “Product Assembly line” game, which involves adding *Components* to assemble a *Product* on a conveyor belt (product moves from right to left across the screen). For a basic game this can involve simply dragging (see supplied code – section 6) a single type of component to the moving product (a specific number of times) before the Product reaches the edge of the screen. For a better grade, different types of *Component* must be added to each of a number of moving Products and the components may also be moving on the screen. The mouse should be used to drag the components to the product being assembled (code is provided, see section 6). 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 –

* “Lunch boxes” Product is a lunch box moving across the top of the screen. A Food item(s) (e.g. a sandwich) appears somewhere on the screen and each lunchbox needs 3 sandwiches to be added before it gets to the left hand edge.
* “Pizza Factory”. Pizza(s) appears and moves right to left across bottom of screen and must be completed before it reaches left hand edge. Topping items fall from top of the screen, can be collected by dragging them to the pizza. Toppings could be of one or more different types. Pizza needs 3 toppings to be complete, but more toppings may increase a score.
* “Bicycle factory”. Product will be a box for a bicycle that needs a frame and 2 wheels (and possibly other components) before it reaches the edge of the screen. Multiple bicycles appear on conveyor at once. Frame and wheels need rotating to the correct orientation to fit into the box
* “Cup-Cake factory” : Assemble cakes : base, fondant, cherry in correct order. Components appear with random rotations and must be rotated to the correct orientation before adding to the cake.

In all examples the game would end when a *Product* reaches the edge of the screen which hasn’t been completed. A counter variable (or multiple) can be used for this, or you could have an array/arrayList of product components that have been added.

All the techniques you need to solve this will be covered in the taught sessions: mouse-drag code is provided (see section 6), collision detection between the mouse and product can be used to drop the component on the product. If you want to rotate an image or shape see section 6.

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.

The accompanying **Implementation Report** must be a word document (or pdf) that explains how you *developed* your solution from starting out to the solution you handed in. It must include screenshots (code snippets) of your code at different stages of development and should consist of no more than 500 words. This will be used to ensure that your solution is entirely your own work.

**Optional additions**

* Game levels
* Player lives
* Scoring
* Multiple animated sequences (must differ in sequencing) – e.g. spinning, flashing, explosion etc.

1. **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.
* Implementation Report (MS Word or pdf)

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.

1. **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). You will also need to demonstrate that your solution is a product of your own effort through your Implementation report.

|  |  |
| --- | --- |
| **Base Mark** | **Features Required** (starting at top of 40% band, complete and tick off each criteria working down the list) |
| 40% | **All** of the following to Pass (40%):   * Development of your solution is fully explained via a written **implementation report (<500 words)** which must include screen shots of section of your code at different stages of development to show your complete understanding of your solution. * Product (to be assembled) moves across the screen (right to left) * Minimum 2 classes (Component, Product) * At least one Component object that can be clicked on or dragged (by the mouse) to the Product * Simple working game (comment out any code that causes errors) in Processing. * Something clearly happens when the Component collides with the Product (e.g. Component removed from game, Product is updated) |
| 40-50% | **All** of the above, and some of the following:   * At least 3 Component objects on screen (can be of the same class type) * Working Boolean Collision *function* method(s) : for the Product with the Component * Game ends when an incomplete Product (<3 components added) gets to edge of screen * Component objects use PImage |
| 50-60% | **All** of the above, and some of the following:   * PImage for Components and Product * Splash or game over screen (draw does different things at different times) * An ArrayList (or array) of Component objects * New Components and Products appear as game progresses * Product and Component objects removed from memory at the appropriate time (set to **null**, or removed from array/arrayList) * A 2nd type (class) of Component object, where Product requires different types of components to be completed. |
| 60-70% | **All** of the above, and some of the following:   * Class-inheritance for different Component types (perhaps other classes) * Moving Product changes appearance as Components are added (can be different images for different stages of assembly or draw the Components on top of the product) * File handling – high score(s) saved and read from file |
| 70-80% | **All** of the above, and some of the following   * Animation sequences for when a Component is added to the Product or when a Component is picked up * Exhibits some polymorphism with the array/arrayList of Components (arrayList stores classes of different types which all inherit from a *super* class) * Another class of objects that must be avoided by the player (collision between mouse position and object involves loss of life or game end). An example could be a puddle of water to avoid with the player mouse, or perhaps a moving rat. |
| 80%+ | **All** of the above, and some of the following:   * Polymorphism for most (or all) game entities * Use of an Interface or abstract class * Complex product building, e.g. Rotation of Components necessary (arrow keys) * 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 | Fully Commented showing complete understanding | 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 |

1. **Support, Hints & Help & Getting Started**

See moodle for support details, you can and ask lab staff or support staff (drop-in sessions) for help or verbal feedback about your solution as you are developing it. Also ensure you keep ticking off the marking criteria as you complete them within your solution. Whenever you make a major change (e.g. moving between grade boundaries on the criteria list) introducing consider taking a screenshot of a code snippet for your implementation report.

Draggable Circle code (adapted from <https://happycoding.io/tutorials/processing/input/draggable-circle>)

float circleX; //draggable circle position

float circleY;

float circleDiameter;

void **setup**() {

size(200, 200);

circleX = width/2; //set circle to be in centre of screen

circleY = height/2;

circleDiameter = 50;

}

void **draw**() { //repeat this code 60 times a second – animate things on screen

background(64);

if ( **dist**(mouseX, mouseY, circleX, circleY) < circleDiameter/2) { //mouse is inside the circle

if (**mousePressed**) { //mouse is inside the circle and clicked

fill(64, 256, 64); //colour it bright green and move the circle to mouse position

circleX = mouseX; //change circle position to mouse position

circleY = mouseY;

}

else { //mouse is inside the circle but not clicked

fill(128, 256, 128); //highlight the circle light green but don't move it

}

}

else { //mouse is outside the circle, color it gray

fill(128);

}

ellipse(circleX, circleY, circleDiameter, circleDiameter);

}

Example class that draws a rectangle rotated around its central point (x,y), by an angle passed as a parameter. You could have the angle stored as a member of the class instead

class **Rotateable**

{

float x,y;

int size;

Rotateable(int x, int y, int size)

{

this.x = x;

this.y = y;

this.size = size;

}

//angleRadians is a an angle measured in radians [0..2Pi], see processing.org/reference/rotate\_.html

void renderRotated(float angleRadians) //draws a rotated rectangle

{

rectMode(CENTER);

// imageMode(CENTER); //this version for a PImage

pushMatrix(); //store everything on canvas

translate(this.x, this.y); //move origin to centre of rotation

rotate(angleRadians); //rotate around x,y

rect(0,0, size, size); //draw the rotated object at the new origin

popMatrix(); //put everything back

}

}

**Getting Started**

In week 6 we had the Defenderz lab exercise which included collision detection between objects of different classes – also covered this in the lectures that week. 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 serve as example for starting 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.

1. Adapt the code (above )for dragging a circle, to become a class for a draggable Circle, test it. This can be used as the basis for your components.
2. Add a class to be your Product
3. Add a collision method – detect when a *dragged object* is very close to the *product*
4. Product should show how many components have been dropped on it (various lab exercises made use of the **text** command).
5. Move product right to left (as if on a conveyor belt)
6. End the game when product reaches left hand side and isn’t already complete

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 (on splash screen, mention in report).

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 classes and 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?

Refactor your code – improve it’s readability (see additions and code quality table).