**Department of Computing and Mathematics**

**ASSESSMENT COVER SHEET 2024/25**

|  |  |
| --- | --- |
| **Unit Code and Title:** | 6G4Z0020 Programming |
| **Assessment Set By:** | Dr David McLean |
| **Assessment ID:** | 2 Cwk 50% |
| **Assessment Weighting:** | 50% |
| **Assessment Title:** | Don’t Crash!! Game |
| **Type:** | **Individual** |
| **Hand-In Deadline:** | assessment week (ending Fri 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. If submitting via OneDrive, ensure that your tutors have access to the work. Do not alter your work after the deadline. You should make at least one full backup copy of your work.

**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 any 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):

* **Non-evidenced extension**: 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. For this assessment, the self-certification extension is 2 days.
* **Evidenced extension**: 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 reassessment 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-life/student-case-management/guidance-for-students/student-code-of-conduct) and [Academic Misconduct Policy](https://www.mmu.ac.uk/legal/policies/misconduct-policy-24-25#:~:text=Any%20attempt%20by%20a%20student,it%20may%20have%20taken%20place.). Poor referencing or submitting the wrong assignment may still be treated as plagiarism. If in doubt, seek advice from your tutor.

**A black background with a black square

Description automatically generated with medium confidenceAs 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.**

**Use of generative AI**

*The use of generative AI is permitted in this assessment, so long as it is used in accordance with the instructions provided in the ‘*[*Are you allowed to use AI in assessments?*](https://rise.mmu.ac.uk/topic/are-you-allowed-ai-in-assessments/)*’ section of the AI Literacy Rise Study Pack (link below).* ***All submitted work must be your own original content.***

*As directed on moodle, you can use AI to check your own code, to help solve bugs or errors, to explain short sections of code you may have found within the course notes or elsewhere and to provide suggestions for design.*

*You may not use it to generate large sections of code and pass that work off as your own. You must be able to fully explain ALL your solution and have a complete and detailed understanding of the solution you submit (see Plagiarism above). You must not use AI to generate any test for your report.*

For any other uses of generative AI, you should also follow the instructions in the ‘Are you allowed to use AI in assessments?’ section of the [AI Literacy Rise Study Pack](https://rise.mmu.ac.uk/courses/ai-literacy/) or speak to your tutor. All submitted work must be your own original content.

**If you are unable to upload your work to Moodle**

If you have problems submitting your work through Moodle, you can send your work to the Assessment Management Team using the [Contingency Submission Form.](https://manmetuni.cherwellondemand.com/CherwellPortal/AssistDL/SAMLlogin/One-Step/DLARConSub) Assessment Management will then forward your work to the appropriate person for marking. If you use this submission method, your work must be sent **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 the [Undergraduate Assessment Regulations](https://www.mmu.ac.uk/legal/policies/undergraduate-assessment-regulations-24-25) on the [Assessments and Results information pages](https://www.mmu.ac.uk/student-life/course/assessments)

|  |  |
| --- | --- |
| 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: | 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, Getting Started, Hints & help
6. Index of techniques (finding concepts on moodle)

# 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 relevant to solving all types of coding problems and are industrially relevant. A completed application can be used as evidence of your problem solving abilities and coding skills for your CV and future placement and career interviews (see <https://www.itjobswatch.co.uk/jobs/uk/degree.do> “Top 30 Co-occurring Skills and Capabilities” for a list of the most common key-phrases currently appearing in job adverts – note “problem solving” and “analytical skills”, which refer to coding and design). 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 Development Report**

You must design and Implement in **Java** **Processing** , a 2-D “Avoid the obstacles” game, where the player stays in the **centre** of the screen, pressing the LEFT key moves all obstacles to the RIGHT, pressing the UP key moves obstacles DOWN etc. Obstacles can also have their own movement across the screen (speedX and speedY). You can choose any game scenario that fits this description (if unsure ask a lab tutor). Your code must contain a minimum of 1 class (Obstacle), though 2 or more classes would be more appropriate and classes can be added as necessary. UP, DOWN, LEFT, RIGHT keys should move ALL obstacles on the screen, relative to the player which stays at the center.

Some possible example game concepts –

* “Ball and boxes”. Player is a simple ball (middle of the screen) which avoids boxes (obstacles)
* “Asteroid Field”. Player is a spaceship (middle of the screen) avoiding asteroids and space junk.
* “Dodge Ball” Player is a person (middle of the screen) avoiding balls coming from different directions.
* “Dodge-Ems”. Player is a car (middle of the screen) trying to avoid being hit by other traffic coming from different directions and possibly collect fuel/spare parts to keep playing.
* “House-Fly” : Player is a fly (middle of the screen) avoiding fly swats, slippers and other household projectiles coming from random directions. Collectible static objects (jam pot, sugar) also appear.

In all examples the game would end when the player collides with an obstacle (or runs out of lives/damage).

All the techniques you need to solve this have/will be covered in the taught sessions, and an [index of concepts is provided](#index) which shows which week on moodle it was covered in. If you want to rotate an image or shape see [section 5 Hints](#Hints) of this specification.

There are no marks available for the quality of the graphics, just for the coding techniques used, so leave any images (gif, jpg etc.) until later in your application development. Start simply with ellipses, rectangles etc. and add images later ([see Graphics](#Graphics)).

The accompanying **Development Report** must be a word document (or pdf) that explains HOW (the order) you *developed* your solution, from starting out to the solution you handed in. It ***must include*** short screenshots (code snippets) from your code at different stages of development. It should state where in your code (tab name/method name or line numbers) that you included key features form the different grade boundaries (e.g. boolean collision function, array, 2nd class type, inheritance, polymorphism etc.) (and should consist of no more than 500 words (be concise). This will be used to help ensure that your solution is entirely your own work.

**Optional additions**

* Player lives
* Scoring
* Game levels
* Scrolling background (difficult)
* Multiple animated sequences (should 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.
* Development 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 60%, all features from the previous bands (40-50),(50-60) must be included within your code, including some features in the 60-70 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 Development Report. Please remember that we have a huge volume of these to mark (expected 600+) so ensure its obvious which features you have included within the first 2 mins of playing the game and stick rigidly to the criteria in the table below.

|  |  |
| --- | --- |
| **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 **Development report (<500 words)** which must include screen shots of code snippets and reference your code to show where you added key features thus demonstrating an understanding of your solution. * Simple working application (comment out any code that causes errors) * Player stays in **centre of screen** * Use of at least one **Class** (Obstacle or Player) * **Keyboard arrow cluster**, shifts at least one **obstacle(s) in opposite direction** to key pressed, e.g. UP key – causes all obstacles to shift down a bit. |
| 40-50% | **All** of the above, and some of the following:   * Minimum 2 classes (Obstacle, Player) * Something obvious happens when Player object collides with Obstacle object (e.g. print a message to console, game stops, loss of life, remove obstacle or player from game) * At least 2 Obstacle objects on screen (can be of the same **Class** type) * Working **Boolean** Collision *function* method(s) for the Player with the Obstacle Class i.e. you should pass an obstacle as a parameter to the Player class collision method which returns true if the player and obstacle are within a certain distance ([dist](https://processing.org/reference/dist_.html) command – follow link). |
| 50-60% | **All** of the above, and some of the following:   * Obstacles have their own movement (when no key is pressed), moving across the screen in any direction. * Obstacle objects use PImage * An ArrayList (or array) of Obstacle objects * New Obstacle objects(s) appear as game progresses * Obstacle objects removed from memory at the appropriate time (set to **null**, or removed from array/arrayList) |
| 60-70% | **All** of the above, and some of the following:   * A 2nd Class (different type) of Obstacle object (you can also add Collectable object class) * Appropriate use of **Class-inheritance** for different Class types * Player changes appearance dependent on arrow key pressed (apparent direction of travel) player render method chooses an appropriate image. |
| 70-80% | **All** of the above, and some of the following   * Animation sequence(s) for Player or Obstacle objects (use of sequence of PImage) * Exhibits some **polymorphism with the array/arrayList** of Obstacles (arrayList stores classes of different types which all inherit from a *super* class) * File handling – high score(s) saved and read from file * Refactored, maintainable code |
| 80%+ | **Most** of the above, and some of the following:   * Polymorphism for most (or all) game entities * Use of an **Interface or abstract class** * Collision animation sequence, e.g. an explosion. An explosion (or similar) class which shows an animation sequence over a short time period, while the game continues. |

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.** | | | |
| Meaningful Variable names:   * Mostly * All | * Score * gameOver screen * lives * splash screen * game levels | Classes:   * Appropriate, * Well-structured * Members * Suitable methods * Function methods – report status | Constructor(s):   * Single * Multiple |
| Well factored – procedures/parameters   * Clear Names * No duplication * Easily read * Concise efficient code | constants - appropriate use  Enum set e.g. game modes | Low class coupling:  Not reliant on global variables  Re-useable classes for future application | Public/Private |
| Well-structured code:   * Mostly * All * 1 Tab per class * Indentation * Intuitive order | comments : code reads as English, comments where necessary |  | Switch case – e.g. control of draw() |

1. **Help**

**5.i. Support**

See moodle for support details (Teaching team) , 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 (from the top down) as you complete them within your solution. Whenever you make a major change (e.g. moving between grade boundaries on the criteria list) consider taking a screenshot of a code snippet for your implementation report.

**5.ii Getting Started**

Watch the [**GettingStarted**](https://moodle.mmu.ac.uk/mod/kalvidres/view.php?id=4432317) video(s) on moodle (see Assessment block). In week 6 Multiple Classes, we had the Defenderz lab exercise and a webinar covering a simple multiclass game. 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. Create a class for your Obstacles, it should be able to display itself at its current position, but not yet move. Create an instance and get it to appear on the screen
2. Add a [keyPressed()](https://processing.org/reference/keyPressed_.html) event under the draw() event to detect the arrow keys and tell your obstacle to move, you could have 4 different move methods or better still pass a value as a parameter to an obstacle move method, (see [**GettingStarted**](https://moodle.mmu.ac.uk/mod/kalvidres/view.php?id=4432317) video on moodle).

Check your obstacle moves a little in the opposite direction to the key that was pressed

1. Add a Player class, draw itself in the **centre** of your screen, just use a simple shape for now and get it to appear on the screen.
2. Create (at least) another instance of the obstacle class and get it on the screen, moving with keypresses as before.
3. Add a Boolean collision method to either your Player or Obstacle class, calling it from within draw. Test it – check it works by printing a message to the console window (see Getting started video).
4. Now add movement behaviour to the Obstacle class so that obstacles move even when no key is pressed. A **move** method (and perhaps an **update** method) should allow obstacles to move across the screen (and wrap around to come back on the screen on other side).

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 (if not break into separate methods)? Is all the information it needs passed as parameters (or already in the class members), avoiding reliance on global variables?

Refactor your code – improve its readability (see additions and code quality table).

**5.iii Hints & Help**

An example class that draws a rectangle and vertical line, rotated around its central point (x,y), by an angle (degrees) passed as a parameter. You could have the angle stored as a member of the class instead. The class includes two useful function methods that will calculate the rotation angle based on a speed vector, or a point the shape appears to be heading towards.

class Rotateable

{

int x,y;

float speedX, speedY;

int size;

Rotateable(int x, int y, int size, float dx, float dy)

{

this.x = x;

this.y = y;

this.size = size;

this.speedX = dx;

this.speedY = dy;

}

//draws a rotated rectangle see processing.org/reference/rotate\_.html

void renderRotated(float angleDegrees)

{

float angleRadians = radians(angleDegrees);

rectMode(CENTER);

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

pushMatrix(); //store coords of everything on canvas

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

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

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

line(0,0, 0,-size);

popMatrix(); //put everything back, including the origin back at top left corner

}

//calculate angle(degrees) for rotate command from speedX,speedY values

float headingToRotationAngle()

{

return headingTo(x+(int)(100\*this.speedX), y+(int)(100\*this.speedY));

}

//calculate angle(degrees) for rotate command for heading towards a distant point

float headingTo(int towardsX, int towardsY)

{

float xDist = towardsX-this.x;

float yDist = towardsY-this.y;

float degrees=0.0;

if (yDist<0)

degrees = degrees( atan( (xDist)/(-yDist) ));

if (yDist>0 )

degrees = 180.0+degrees(atan( (-xDist)/(yDist) ));

return degrees;

}

}

**5.v. Graphics**

There are many sources of images (Paint 3D) or online editors (e.g. <https://www.piskelapp.com/p/create/sprite> ) 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). PNG images can be created with transparent backgrounds.

1. **Index of Coding Techniques**

where on moodle you can find each of the coding concepts required for the assessment criteria, these will become available as the course progresses

|  |  |  |
| --- | --- | --- |
| **Base Mark** | **Coding concepts required** | **Where on Moodle,** links take you to week header – scroll down to indicated saction. Note :Video tutorials at the end of each weekly |
| 40% | **All** of the following to Pass (40%):   * Minimum 1 class (e.g. Obstacle, Player) * Keyboard arrow cluster | **Week 5** [Classes & Objects](https://moodle.mmu.ac.uk/course/view.php?id=183727#section-10) : webA – Intro to using a class, lab and see video solutions **Week 6** [Multiple Classes](https://moodle.mmu.ac.uk/course/view.php?id=183727#section-11), see Web A : Space Invaderz & video support and solutions  [Getting started video](https://moodle.mmu.ac.uk/mod/kalvidres/view.php?id=4432317)  **Week4** [Events & Animation](https://moodle.mmu.ac.uk/course/view.php?id=183727#section-9), & Pong game lab exercise  [keyCode / Reference / Processing.org](https://processing.org/reference/keyCode.html) |
| 40-50% | * Minimum 2 Classes (Player,Obstacle) * Obstacle collides with the Player * Working Boolean Collision *function* method(s) * Obstacle objects use PImage | **Week 6** [Multiple Classes](https://moodle.mmu.ac.uk/course/view.php?id=183727#section-11), see Web A : Space Invaderz, also [week8 webA](https://moodle.mmu.ac.uk/course/view.php?id=183727#section-13) (birds and plane game)  **Week 6** [Multiple Classes](https://moodle.mmu.ac.uk/course/view.php?id=183727#section-11), Web A : Space Invaderz, also [[week8 webA](https://moodle.mmu.ac.uk/course/view.php?id=183727#section-13) (birds and plane game) webA](https://moodle.mmu.ac.uk/course/view.php?id=172004#section-10) (flying birds demo)  **Week 4 :** [web 4B](https://moodle.mmu.ac.uk/course/view.php?id=183727#section-9) (flying bird) and later weeks  [loadImage() / Reference / Processing.org](https://processing.org/reference/loadImage_.html) |
| 50-60% | * Obstacles have their own movement * An ArrayList (or array) of Obstacle objects * objects removed from memory (set to **null**, or removed from array/arrayList) | **Week 5** [Classes & Objects](https://moodle.mmu.ac.uk/course/view.php?id=183727#section-10) : webA – Intro to using a class, lab and see video solutions **Week 6** [Multiple Classes](https://moodle.mmu.ac.uk/course/view.php?id=183727#section-11), see Web A & Video support and solutions  [Week7 : lab](https://moodle.mmu.ac.uk/course/view.php?id=183727#section-12), web B, [Week8 webA](https://moodle.mmu.ac.uk/course/view.php?id=183727#section-13) (birds and plane game)  [Week8 webA](https://moodle.mmu.ac.uk/course/view.php?id=183727#section-13) (birds and plane game) , [Week7 : lab](https://moodle.mmu.ac.uk/course/view.php?id=172004#section-9), web B |
| 60-70% | * 2nd Class (different type) of Obstacle * Class-inheritance * Player changes appearance dependent on arrow key | [Week 9 : Inheritance](https://moodle.mmu.ac.uk/course/view.php?id=183727" \l "section-14)  [See Hints and help](#Hints) or PImages : **Week 4 :** [web 4B](https://moodle.mmu.ac.uk/course/view.php?id=172004#section-6) (flying bird) and later weeks |
| 70-80% | * Animation sequences * File handling – high score(s) saved and read from file * polymorphism with array/arrayList | Week 4 : [web 4B](https://moodle.mmu.ac.uk/course/view.php?id=183727#section-9) (flying bird) and [Week5, web B](https://moodle.mmu.ac.uk/course/view.php?id=183727#section-10) (Walker demo)  [Week11, Web A,](https://moodle.mmu.ac.uk/course/view.php?id=183727#section-16) or [loadStrings() / Reference / Processing.org](https://processing.org/reference/loadStrings_.html)  [Week 10 , Web A](https://moodle.mmu.ac.uk/course/view.php?id=183727#section-15) |
| 80%+ | * Use of an Interface or abstract class | [Week 10 , Web A](https://moodle.mmu.ac.uk/course/view.php?id=183727#section-15) (abstract class), WebB (Interface) |