Part Two: The Belly of the Beast

创建者：Nick Patrikeos
由 Sienna Archer 最后更新于 3月 15, 2023 • 80 人已查看

This page contains the tasks you'll need to complete for this assignment, and how you'll be assessed.

📜 For this assignment, you will need to make **two sets of blog posts**:

- One which is a **pair-completed blog post**, which will contain your answers to questions in Task 1, designs for Task 2 and completion of Task 3.

  Here is a template for your pair blog post

- One is your **individual blog post/s**. Similar to Assignment I we recommend you write 2-4 blog posts (they don't have to be long!) over the course of the assignment.

1. Getting Started

You can access your pair repository via the following URL:

```
https://gitlab.cse.unsw.edu.au/COMP2511/23T1/teams/YOUR_TEAM_NAME/assignment-ii
```

Replace `YOUR_TEAM_NAME` with your team name (e.g. `M18A_ALPHA`)

Watch the following video which goes through approaching the codebase and some VSCode tips and tricks to help you out.

COMP2511 Assignment II: Dungeonmania - The Belly of the Beast

> Tip: Explore and code along with the video to get the most out of it!

2. Tasks

Task 1) Code Analysis and Refactoring (45 marks) 🛠️

In this task, you will need to analyse the design of the monolith, including the patterns and smells present in

the implementation, and apply refactoring techniques discussed in the course in order to improve the quality of the code.

🖌️ Put all of your answers to the following theory questions in your **pair blog post**.

For each refactoring change, **before you start coding** take some time to plan and write up a design in your pair blog post. This should include:

- What fields/methods you will need to add/change in a class
- What new classes/packages you will need to create

After you finish coding, make a Merge Request into the `master` branch of your repository with the following:

- A meaningful MR title that encompasses the changes
- A brief description which outlines the changes being made
- Make sure to keep the MR as small as possible - don't make any extra changes that aren't absolutely necessary. Try to keep the number of files the changes touch to a minimum.
- Make sure all the Continuous Integration checks (regression tests, linting, coverage) remain passing.

Your partner will need to **code review** and either:

- Leave comments in the MR with things to fix (you can do code reviews sync or async), requiring you to **iterate on the MR** and **resubmit for review**; or
- **Approve** the MR.

Once the MR is approved, copy and paste the link to your MR into your blog post.

🌀 Make sure to do this for each part of the question **if you don't you will lose marks**.

a) From DRY to Design Patterns (5 marks)

i. Look inside `src/main/java/dungeonmania/entities/enemies`. Where can you notice an instance of repeated code? Note down the particular offending lines/methods/fields.

ii. What Design Pattern could be used to improve the quality of the code and avoid repetition? Justify your choice by relating the scenario to the key characteristics of your chosen Design Pattern.

iii. Using your chosen Design Pattern, refactor the code to remove the repetition.

b) Observer Pattern (5 marks)

Identify **one place** where the **Observer Pattern** is present in the codebase, and outline how the implementation relates to the **key characteristics** of the Observer Pattern.

implementation relates to the **key characteristics** of the Observer Pattern.

c) Inheritance Design (5 marks)

Currently, there is a significant flaw in the inheritance structure of entities.

Consider the following three methods in the `Exit` entity class.

```
@Override
public void onOverlap(GameMap map, Entity entity) {
    return;
}

@Override
public void onMovedAway(GameMap map, Entity entity) {
    return;
}

@Override
public void onDestroy(GameMap gameMap) {
    return;
}
```

i. Name the code smell present in the above code. Identify all subclasses of `Entity` which have similar code smells that point towards the same root cause.

ii. Redesign the inheritance structure to solve the problem, in doing so remove the smells.

d) More Code Smells (5 marks)

The previous engineering team has left you with the following note:

> *Collectable entities are a big problem. We tried to change the way picking up items is handled, to be done at the player level instead of within the entity itself but found that we had to start making changes in heaps of different places for it to work, so we abandoned it.*

i. What design smell is present in the above description?

ii. Refactor the code to resolve the smell and underlying problem causing it.

e) Open-Closed Goals (5 marks)

Look inside the `goals` package at the code provided.

i. Do you think the design is of good quality here? Do you think it complies with the open-closed principle? Do you think the design should be changed?

ii. If you think the design is sufficient as it is, justify your decision. If you think the answer is no, pick a suitable Design Pattern that would improve the quality of the code and refactor the code accordingly.

f) Open Refactoring (20 marks)

Make any other refactoring improvements you see fit to the codebase. This can include resolving Design Smells, using Design Patterns discussed or any other general improvements to the health and quality of the code.

Some areas and questions you can consider:

- Look for violations of the Law of Demeter/Liskov Substitution Principle;
- The effects of potions (invisibility and invincibility) has been implemented using a State Pattern. However, the State Pattern hasn't been used particularly effectively here and as a result, there is poor design.
- The current implementation of buildable entities contains a significant amount of hard coding. Think about how you can improve this.

The above list isn't exhaustive; there are plenty of other areas to improve the quality of the code.

⚠️ Don't make solutions to problems that don't exist yet! :) Avoid over-engineering or over-abstracting in places where you *might* want to improve the design for some future change in requirements - instead improve the design of the current system. This will inherently make your software open-closed.

You'll also want to split this task into one MR for each refactoring change you make.

Task 2) Evolution of Requirements (50 marks) 👽

In this task, you and your partner/group will need to complete **Part A - Microevolution** and the same number of the following tasks as there are members in your group:

b) Bosses (10 marks)

d) Swamp Tile (10 marks)

c) Sunstone & More Buildables (15 marks)

g) Dungeon Generation (15 marks)

e) Time Travel (20 marks)

f) Logic Switches (20 marks)

E.g. If there are two of you, you need to pick two of the above tasks, if there are three of you, you pick three. Each task has a different mark allocation based on its difficulty. You should pick tasks you feel comfortable completing. If you complete extra tasks, that is fine - we will take your performance in the best N tasks, where N is the number of people in your team.

If you are in a group of 3 your mark will be out of 65.

🔗 Software Delivery - Task Lifecycle

For each part of this task, you will need to undertake the following process:

1. **Requirements Engineering**. Analyse the task requirements, including the technical and product specifications. If you need to, make some assumptions and document these in your pair blog post.
2. **Detailed Design**. In your pair blog post, plan out a detailed design for the task. This should include:
   - What fields/methods you will need to add/change in a class
   - What new classes/packages you will need to create
3. **Design Review.** Have your partner review the design, and go back and iterate on the design if needed.
4. **Create a Test List**. Once the design is approved, write a **test list** (a list of all the tests you will write) for the task. Map out each of the conceptual cases you want to test. This can be written in your blog post, or if you want make function stubs for JUnit tests and put up a Merge Request (link in your blog).
5. **Test List Review**. Have someone else in your team review the test list to make sure the test cases all make sense and cover the input space.
6. **Create the Skeleton**. Stub out anything you need to with class/method prototypes.
7. **Write the tests**, which should be failing assertions currently since the functionality hasn't been implemented.
8. **Development**. Implement the functionality so that your tests pass.
9. Run a **usability test** (check your functionality works on the frontend).
10. Where needed, **refactor your code** to improve the style and design while keeping the tests passing.
11. Put up a merge request with your changes into `master`. The CI should be passing. The merge request should have a meaningful title and contain a description of the changes you have made. In most cases you should just be able to link them to your design/test list blog.
12. **Code Review** from your partner, iterate where needed then they should approve the MR.

Feel free to split tasks further where you see fit, especially for larger tasks (e.g. Time Travel). Keep your Merge Requests small and make the iteration cycle short, incrementally building your MVP as you go.

If you've done Task 1 well and have a nice healthy codebase, this task should be relatively straightforward! If you're finding parts of this task difficult to integrate with the existing design, that's probably a sign you need to do some more refactoring :)

You will need to write tests for the new functionality in an appropriate section of the test suite.

a) Microevolution - Enemy Goal (10 marks)

In this section you will need to make a series of small-scale changes and additions to the code based on the following new requirements.

The following new goal has been introduced:

- Destroying a certain number of enemies (or more) AND all spawners;

Other goal rules, including rules of conjunction/disjunction and exits must be completed last, still apply.

b) Bosses (10 marks)

In this task, you need to implement the following new entities.

Bosses are moving entities which are harder to defeat/conquer than normal enemies.

Entity	Image	Description
Assassin		Assassins are exceptionally powerful mercenaries which deal significantly more damage. When bribing an Assassin, there is a certain chance that the bribe will fail; the gold will be wasted and the Assassin will remain hostile. Battles still do not occur with an Assassin when the player is invisible.
Hydra		Hydras are generally considered to be a special creatures similar to Zombies. Hydras are limited by the same movement constraints as Zombies. In each round, when a hydra is attacked by the player, there is a certain chance that its health will increase rather than decrease by the given amount, as two heads have grown back when one is cut off.

c) Swamp Tile (10 marks)

c) Swamp Tile (10 marks)

As part of this, you will need to extend your solution to accommodate the idea of a **swamp tile**. These are tiles that have an `x` and `y` position and remain fixed throughout the entire game. They slow the movement of all entities through them, except for the player and allies adjacent to the player. Each swamp file has a movement factor which is a multiplying factor of the number of ticks it takes to traverse the tile. For example, let us say the movement factor for a swamp tile is 2:

- Tick 1: Move onto the swamp tile;
- Tick 2: Stuck on the swamp tile;
- Tick 3: Still stuck on the swamp tile;
- Tick 4: Move off the swamp tile.

Entity	Image
Swamp Tile	

d) Sun Stone & More Buildables (15 marks)

i) a) Further Collectable Entities

In this task, the following collectable entities need to be added:

Entity	Image	Description
Sun Stone		A special form of treasure, hard and treasuable. It can be picked up by the player. Can be used to open doors, and can be used interchangeably with treasure or keys when building entities. But it cannot be used to bribe mercenaries or assassins. Since it is classed as treasure it counts towards the treasure goal. When used for opening doors, or when replacing another material such as a key or treasure in building entities, it is retained after use.

i) b) Further Buildable Entities

In this task, the following buildable entities have been added:

Entity	Image	Description
Sceptre		Can be crafted with (1 wood OR 2 arrows) + (1 key OR 1 treasure) + (1 sun stone). A character with a sceptre does not need to bribe mercenaries or assassins to become allies, as they can use the sceptre to control their minds without any distance constraint. But the effects only last for a certain number of ticks.
Midnight Armour		Can be crafted with (1 sword + 1 sun stone) if there are no zombies currently in the dungeon. Midnight armour provides extra attack damage as well as protection, and it lasts forever.

e) Dungeon Generation (15 marks)

This task involves modifying the frontend code; if you're interested in learning a bit of frontend and trying TypeScript then have a go!

In this extension, instead of specifying an existing dungeon to play, players can choose specify a dungeon to be automatically generated when creating a new game.

As part of this, you will need to be able to automatically generate dungeons. Furthermore it's important that you have an *exit* at the end position and that you have exit goals setup for this created dungeon.

3.12.1 Generating a Dungeon - Randomized Prim's Algorithm

You will need to generate dungeons according to the following maze generation algorithm (which is just a randomised version of Prim's).

› Algorithm

3.12.2 Frontend Code

In order to implement the frontend for this task you'll need to modify the TypeScript code inside `client/src`.

> More Guidance

f) Time Travel (20 marks)

i) Time Turner

This part of the extension includes the following new entity:

Entity	Image
Time Turner	

If the player has collected a time turner, then two rewind buttons will appear on the frontend. When clicked, these buttons move the state of the game back one tick and 5 ticks respectively and "transport" the current player back to those game states in a time travelling fashion.

ii) Time Travelling Portal

This part of the extension includes the following new entity:

Entity	Image
Time Travelling Portal	

If a player travels through a time travelling portal, they end up on the same square as the portal, except the dungeon state is that of 30 ticks previously. If less than 30 ticks have passed, then the dungeon state is simply the initial dungeon state.

iii) Time Travel Rules

When a character has time travelled, either by the rewind buttons or via a time travelling portal:

- Their 'older self' still exists in the dungeon as its own entity. If they encounter their older self and are invisible, then nothing happens. If not, then a battle ensues.
- The older self should take the same path as was taken initially, and unless they encounter their 'current

self (the character being controlled), should eventually travel through the time portal and disappear from the map.

- The player's inventory persists across time travelling. This means that if a player picks up a sword then travels through a time portal, the sword remains in their inventory as well as being back on the map available to pick up.
- The older self is in the enemy party and doesn't fight the enemies.
- Mercenaries and assassins follow the current player, not the older player
- The behaviour when time-travelling occurs more than once in a single game instance is undefined.
- Only the player can travel through time travel portals.
- The older player should still collect items and play out all `tick` and `interact` movements with those items as they did before.
- Note that usage of a time turner (a `rewind` operation) is not the same as travelling through a time portal - the latter takes up a tick, whereas the former does not.
- The following diagram should help you understand the state transitions for time travel.

> Tick Diagram: Time Travel

> Implementation Hint

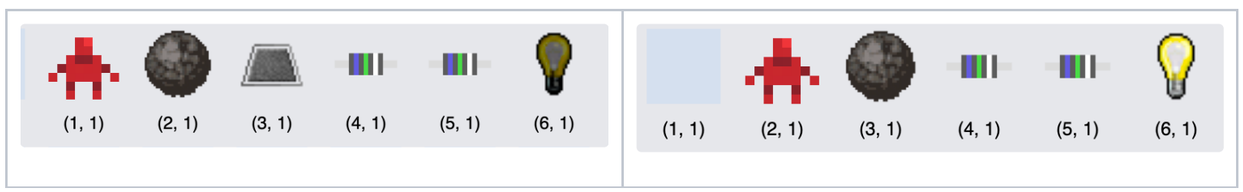
g) Logic Switches (20 marks)

There are three new entities in this extension:

Entity	Image	Description
Light Bulb		Light bulbs cannot be collected, but can be lit up by satisfying the logical condition. They are always created as a logical entity. Light bulbs will always be created off.
Wire		Wires cannot be collected, but form part of a circuit as a conductor and are connected to a switch and can also connect to any entity that can be activated via switches (Light Bulb, Switch Door). Wires are not themselves logical entities. Any moveable entity can walk onto a wire.
Switch Door		Switch doors are always created as a

		logical entity, and can be opened when their logical condition is satisfied. They cannot be opened with a key. The door should only remain open as long as its condition for opening is still true.

If a switch cardinally adjacent to a wire is activated, **all the other logical entities** cardinally adjacent to the wire are activated. This allows for the creation of dungeons with logic circuits. For example:



v) a) Logical Entities

Entities which interact via switches can become “logical” entities with a specified configuration value:

- Light Bulbs (see above)
- Switch Doors (see above)

Bombs also now have the option to be created as logical entities with a specified configuration value. Bombs which have this attribute will only explode when their logical condition is fulfilled. Bombs created without this value will interact with switches in the same way as they do in the MVP, and do not interact with other logical entities or wires.

Switches are **not logical entities**, but instead activate **cardinally adjacent logical entities, also known as conductors** when switched on. When placed next to another already activated entity, they behave akin to wires in a circuit **if they are switched on**. Light bulbs and switch doors **do not act as conductors**.

All switches will be created in inactive state as well as all the logical entities.

All other entities cannot be part of the circuit. They can be placed adjacent to the logical circuit but don’t have any effects.

v) b) Logical Rules

Entities will be logical according to one of the following rules:

- AND - the entity can be activated if there are 2 or more cardinally adjacent activated conductors. If there are more than two conductors, all must be activated
- OR - the entity will be activated if there is 1 or more cardinally adjacent activated conductor
- XOR - the entity will be activated if there is 1 and only 1 cardinally adjacent activated conductor

- XOR - the entity will be activated if there is 1 and only 1 cardinally adjacent activated conductor

- CO_AND - the entity will only be activated if there are 2 or more cardinally adjacent activated, which are **all activated on the same tick**, e.g. if a switch activates two wires that are both cardinally adjacent to a logical entity with the CO_AND condition, it should be activated.

Task 3) Investigation Task ⁉️ (5 marks core + 5 marks bonus)

How confident are you that your software satisfies the requirements?

You will need to review the codebase and specification to ensure that all the requirements are correctly implemented, both of the MVP implementation (provided by the previous engineering team) and your extensions to the monolith. If you discover that the software fails to correctly implement any of the requirements, explain how you came to your conclusions in your blog post and outline the steps you took to address the problems in your pair blog post.

This task is intentionally meant to challenge your ability to think outside the box, explore and investigate.

3. Things to note

🧭 Git Practices

- We will not be assessing your commit messages, though for the sake of good practice you should write meaningful commits.
- Instead, when you merge a branch into master select the **Squash Commits** option. This will squash all of your branch-specific commits into a single commit during merge. Make sure to **uncheck** the delete branch option so that your branch is preserved.
- The `master` branch should always remain stable - the pipeline should always be passing. You are welcome to comment out some of the example tests we have provided until you can get them working locally.
- When putting up Merge Requests, all changes in the diff should be relevant to the task which they relate to (not polluted with irrelevant changes, as this makes it difficult for reviewers);
- Code reviews should occur as comments on Merge Requests - if you are doing the code review synchronously leave comments on the MR as your minutes/action items

👽 Assumptions

As you develop your implementation of the Tasks you will undoubtedly come across scenarios where the behaviour is not defined by the rules of the specification. This is a complicated assignment if we

the behaviour is not defined by the rules of the specification. This is a complicated assignment if we defined every single scenario, the spec would go on forever. Unlike previous courses this course is about design, we are not trying to catch you out by testing every single edge case imaginable. The autotests are there to check you followed the spec and check you did the work, not to catch you out on niche edge cases.

Here are the steps you should follow if you are unsure about something in the spec:

1. Double check the spec, do a ctrl/command-f to check it isn't mentioned elsewhere, in a lot of situations this will be the case.
2. Check the Approved Assumptions document, all of the approved assumptions from last term are there and new ones will be added if they come up.
3. Make a post on the Assumptions Megathread in the forum **(please search first)** asking the Course Staff whether you are able to make an assumption about the behaviour in this case.
   We will either:
   a. Approve the assumption and add to the Approved Assumptions page
   b. Update the specification if appropriate, or
   c. Respond explaining how the behaviour is defined in the scope of the specification.

Any ambiguities/assumptions that we have listed as approved **we will not be testing in automarking**.

✅ **Test Design**

When it comes to writing tests for the new features, you should write **functional tests on the controller**.

All of the existing tests in the monolith are written in this way, and you can structure your tests similarly.

🎃 **Dungeon Map Helper**

We've provided a dungeon map helper for you here to assist with visualising and creating test maps. Please note it may contain some bugs! 🐛

4. Design

4.1 Individual Blogging

As well as your pair blog, you will need to complete individual blog post/s each week. These can be more **reflective** than **descriptive**, since your design decisions should be documented in your pair blog post. Each week you can give a brief summary of how things are going, what challenges you are facing and what you want to change next week. In the final blog post you will also need to briefly write about how you think you went overall in the assignment, the challenges you faced and what you learned completing the tasks.

You will also need to put in **links to all your Merge Requests**.

We will use these blog posts to determine individual marks in the case where pair contribution is not equal. You can make your individual blog accessible to staff only if you like.

⚠️ Put links to all your **individual** posts and your **pair post** inside `blog.md` in your repository.

4.2 Interview

During your **Week 10 lab**, you and your partner will have a short (7 minutes) interview with your tutor or lab assistant regarding the design of your solution.

You will not be directly assessed on this interview - it is an opportunity for your tutor to give you some initial feedback on your design, and demonstrate an understanding of your refactoring changes, extensions and mystery task if you completed it.

You will not be able to take any notes into the interview, as you should be familiar enough with your solution to speak to it in depth. Your tutor will lead the interview and give you prompts for you to talk about your design. You can also consider the writing of your blog posts as preparation for the interview.

5. Assessment

5.1 Marking Criteria

Task	Subtask	Criteria
Task 1 🛠️	a) From DRY to Design Patterns (5 marks)	• Has the smell been pinpointed? • Has a suitable Design Pattern been selected and justified? • Was the pattern well implemented?
	b) Observer Pattern (5 marks)	• Was the pattern identified, with key characteristics in the implementation described in sufficient depth?

	c) Inheritance Design (5 marks)	• Was the design smell identified? Were all the relevant classes identified? • Was the design flaw resolved?
	d) More Code Smells (5 marks)	• Was the design smell identified? • Was the design flaw resolved?
	e) Open-Closed Goals (5 marks)	• Is the stance justified according to the design? • If applicable, was any refactoring done to improve the design?
	f) Open Refactoring (20 marks)	For 5 marks, **some** additional smells/flaws are identified and resolved. For 10 marks, **listed** additional smells/flaws (or equivalent) are identified and resolved. For 15 marks, **all** additional smells/flaws are identified and resolved. For 20 marks, **design improvements** are made to the codebase beyond simple smells and flaws.
Task 2 👽	For each subtask completed; All of: a) Microevolutions - Enemy Goal (10 marks) Number of team members of: b) Bosses (10 marks) c) Swamp Tile (10 marks) d) Sunstone & More Buildables (15 marks) e) Dungeon Generation (15 marks) f) Time Travel (20 marks) g) Logic Switches (20 marks)	**Software Correctness (40% of the marks for the subtask)** This section will be automarked. **Software Design (50% of the marks for the subtask)** • Is the design seamlessly integrated into the existing infrastructure? • Does the design adhere to principles discussed in the course? • Does the design contain any smells? **Software Testing (10% of the marks for the subtask)** • Have functional tests on the controller been written? • Do the tests cover a range of cases? • Are the tests well designed?
Task 3 ⁉️	5 marks core + 5 marks bonus	How well was the investigation completed?
Code Hygiene 🪠	We will apply a penalty of up to 5 marks if the following criteria are not maintained	• All regression tests remain passing • Code Coverage remains above 80%

	following criteria are not maintained:	• Code Coverage remains above 80% • The linter remains passing In simpler terms, you must maintain a passing pipeline on the `master` branch of your repository throughout the entire assignment.
Merge Requests 🧭	We will apply a penalty of up to 5 marks if you: • Have Merge Requests that are too large and contain too many changes • Don't link your Merge Requests in your blog post This is to make it easier for your marker to award you for your work. If you stick to the blog template you will be fine here 🙂	
Software Delivery 🚚	We won't assess this directly, but you'll need to show evidence of completing the Task Lifecycle via your pair blog post.	

5.2 Automarking

5.2.1 Autotest Feedback

We will provide you with some **formative autotest feedback** prior to submission. On **Tuesday 11th April** (Week 9) **at 8 am,** we will take the latest commit on your `master` branch and run that against our autotest suite. By **6 pm** that night, we will provide you with a set of results that contain:

- For a subset (a) of our tests, the name of the test, whether you passed or failed it, and if you failed what the error was;
- For another subset (b) of our tests, whether you passed or failed, with all other output redacted. The test name will be redacted to "Hidden test", though the section under which it is testing will be made available.
- The autotest mark you would receive if that was your submission.

There will not be more than one feedback run prior to the deadline for each submission.

In some cases we may update our autotests between the initial feedback run before the deadline and the actual marking of your submissions, as our tests may contain some small errors due to the nature and scale of the project. If this occurs we will post on the forum informing which tests are changing and how they have been changed/corrected.

If you are unsure why you are failing a specific test, review your code and the specification, and if you are still unsure please post on the forum and we will be able to provide you with some support.

5.2.3 Autotest Re-Runs

Following the release of automarks, pairs may request a re-run if the pair has failed more than 30% of the autotests due to **one or two** minor issues in their code. The pair can make a patch of up to 20 lines difference and push this patch to a branch where it will be rerun **at a 20% penalty to the final automark**. Diffs of more than 20 lines need to be approved by the Course Authority. All requests for re-runs must be made via the Course Forum. Pairs that have not failed more than 30% of the autotests, or have failed due to multiple issues / non-minor issues are not elegible for a re-run. If you are not sure whether you are elegible for a re-run please post on the Course Forum.

5.3 Submission

We will take your most recent commit on `master` as your submission during marking. Any commits on `master` after the deadline will be ignored.

Put links to your **pair blog post**, and **both of your individual blog posts** inside `blog.md` in your repository.

The pair blog can be made in either of your personal Confluence spaces. Make sure it is accessible to `cs2511-23t1-staff` for marking (If you just publish it normally this is already the case).

5.4 Late Penalties

This assignment has no late penalty or extension.

Because the vivas need to be conducted in week 10, this assignment cannot be pushed back any further. Any submissions after **Friday, Week 9 at 5pm** will not be marked.

5.3 Extenuating Circumstances

Mark leniency in extenuating circumstances must be approved through either Special Consideration, which needs to be submitted prior to the assignment deadline, or pre-arranged through an Equitable Learning Plan with Equitable Learning Services and the Course Authority. In all cases please email cs2511@cse.unsw.edu.au.

5.4 Other Expectations

While it is up to you as a pair to decide how work is distributed between you, for the purpose of assessment there are certain key criteria all partners must attain:

- Code contribution;
- Non-code contribution;
- Usage of Git/GitLab;
- Participation in the assignment interview;
- Individual blog posts; and

- Academic conduct.

The details of each of these are below.

While, in general, both students will receive the same mark for the assignment, **if you as an individual fail to meet these criteria your final assignment mark will be reduced**.

> ❗ If you believe a your partner is not contributing as they should contribute, you must inform your tutor at the end of that corresponding week.
>
> For example, if your partner has not contributed in Week 5, you need to report this before the end of Week 5. You must not wait beyond this. If you fail to report in time, we may not be able to address the issue and/or apply redistribution of marks.

5.5 Plagiarism

The work you and your partner submit must be your own work. Submission of work partially or completely derived from any other person or jointly written with any other person is not permitted. The penalties for such an offence may include negative marks, automatic failure of the course and possibly other academic discipline. Assignment submissions will be examined both automatically and manually for such submissions.

> ⚠️ Don't look up previous solutions to this assignment on the internet and use them in your submission.

The use of code synthesis tools, such as GitHub Copilot, is not permitted on this assignment. The use of ChatGPT and other generative AI tools is not permitted on this assignment.

Relevant scholarship authorities will be informed if students holding scholarships are involved in an incident of plagiarism or other misconduct.

Do not provide or show your project work to any other person, except for your group and the teaching staff of COMP2511. If you knowingly provide or show your assignment work to another person for any reason, and work derived from it is submitted you may be penalised, even if the work was submitted without your knowledge or consent. This may apply even if your work is submitted by a third party unknown to you.

Note, you will not be penalised if your work has the potential to be taken without your consent or knowledge.

成为第一个添加回复的用户