SENG201 – Software Engineering I
Project Specification
Sports Tournament

For project admin queries:
Matthias Galster — Miguel Morales — Morgan English
{matthias.galster, miguel.morales, morgan.english}@canterbury.ac.nz

For technical support, project help, hints and questions:
Danita Sun — Michael Alpenfels — Harrison Tyson — Jack Fawthorpe
tut201@cosc.canterbury.ac.nz

March 21, 2023

1 Introduction

1.1 Administration

This project is a part of the **SENG201** assessment process. It is worth **30%** of the final grade and it is capped up to 100 marks (i.e., the total marks for it can't exceed 100). This project will be done in pairs (i.e., teams of two students). You must find your own partner and register the team on LEARN. Submissions from individuals will not be accepted. Pairs are not allowed to collaborate with other pairs, and you may not use material from other sources without appropriate attribution.

Plagiarism detection systems will be used on your report and over your code, so do not copy the work of others. Plagiarised projects will be referred to the University proctor for disciplinary actions. Your deliverables must be submitted on LEARN. Students will be asked to demo their project during lab and lecture times.

Actual dates and times are detailed in Section 6.3. The drop dead policy and other penalties are detailed in Section 6.4.

You can find all the project-related information and resources in the "Project" section on LEARN.

1.2 Outline

This project will give you an idea of how a software engineer would go about creating a complete application (which features a graphical user interface) from

scratch. **In this project you will build a game in which you create and manage a sports team. You will be able to draft athletes; buy training equipment to improve their stats; compete against other teams in matches to earn money; take a bye until the next week to rest and refresh the market inventory. You will keep competing and training until the season is complete with the goal to earn the most points you can.**

The idea of the game is slightly open to allow some room for creativity, but please ensure you implement the main project requirements as that is what will be graded. We encourage you to think of the assignment as a *project* rather than a *programming assignment*. Programming assignments often have a clearly defined specification that you can follow step-by-step. However, in practice, software engineering projects require a lot of thinking from the engineers to find out what exactly to build, why, for whom, and how. Therefore, specifications and requirements are often vague and need to be clarified during development.

1.3 Project Management

Adequate planning will help you minimize risks which can negatively impact your project (e.g., falling behind, poor testing). Defining time estimates, clear goals and realistic milestones will give you a much higher chance to succeed. To help you with this, you must record the following data weekly:

- Hours you spent working on the project during the week.
- Activities you carried out during the week.
- If you achieved the goals planned for the week or not.
- The risks you identified or faced during the week.
- How satisfied you are with your and your partner's contribution to the project during the week.

A quiz to record this data is available on LEARN.

You will earn marks by entering the data on time. Similarly, you will lose marks if you fail to regularly record the data, see Section 6.4 for more details.

1.4 Help

This project can get confusing and frustrating at times when your code does not work. This will likely be the largest program you have written thus far, so it is **very important** to break larger tasks into small achievable parts, and then implement small parts at a time, **testing as you go**.

Having a nice tight modular application will help with debugging, so having appropriate classes is a must. If you are having problems, try not to get too much help from your classmates, and instead ask for help from your tutors. You can email them or ask them questions in labs. Always save your work and have

backups, do not assume that the CSSE department will be able to recover any lost data. We invite you to use a Version Control System (VCS) of some kind such as the university's GitLab[1] or a personal GitHub, for more information about Git see Tutorial 2.

2 Requirements

This section describes what your game must do and is written as a set of requirements. At the beginning of your project and to get started, try thinking of each requirement as a separate ticket that needs to be closed **before** others are started, it will help you have code which works and can be built upon, instead of a lot of broken spaghetti code.

Hint: Functionality can be placed in its own package or class. Modularisation is the key, especially when you begin GUI programming.

2.1 Setting Up the Game

When your game first starts it should ask the player to:

1. Choose a team name. The length must be between 3 and 15 characters and must not include special characters.
2. Select how many weeks they would like the season to last (between 5 and 15 weeks).
3. Purchase the starting athletes for your team:
   (a) A team should be a fixed size and have a minimum of 4 athletes.
   (b) Each athlete should have different statistics. Including offence, defence, stamina. Get creative!
   (c) The player shall have reserves available to make substitutions for each position in the team.
   (d) The player shall be able to nickname the athlete (or have a default name)
4. Choose a difficulty
   (a) There must be at least 2 options.
   (b) This should scale how difficult the game is in a noticeable way. (For example how much money the player starts with and/or gets for winning matches, or how hard the opponents will be).
   (c) You may want the harder difficulties to give more points for the same action.
5. Start your season.

[1] `https://eng-git.canterbury.ac.nz/`

2.2 Playing the Main Game

Once the player has finished their setup and drafted their starting team, the main game can begin. The player starts out on week one with a default amount of money, ready to draft new players or improve them with training equipment and other items. The amount of money is up to you (and may be scaled with difficulty). The following options will be displayed to the player:

1. View the amount of money you have, the current week, and the number of weeks remaining.
2. Go to the Club and:
    (a) View the properties of your team. This shall include:
        i. The name of the team.
        ii. The name of each athlete (this may be the default name e.g. John Smith, or a nickname given by the player).
        iii. The properties of each athlete (importantly stamina, offence and defence though all should be viewable).
        iv. It should be clear to the player what positions their athletes are in (if this matters for your chosen sport).
        v. It should be clear which athletes are actively in the team and which are reserves.
        vi. The player should be able to swap athletes on the team with reserves.
        vii. There should be a maximum of 5 reserves.
    (b) View the player inventory, and the current items they have. Each item shall:
        i. Show the effects of the item.
        ii. Allow for the player to use the item on an athlete.
3. Go to the Stadium
    (a) The player should be able to see a small number (3-5) of optional matches they can take on.
        i. The matches should be generated somewhat randomly, but largely influenced by the current week, and optionally the difficulty setting
        ii. The money and points gained for winning a match may scale with the difficulty (e.g. less money on hard, but more points given)
    (b) The player should be able to choose a match to play.
        i. Your team can partake in a match only once.
        ii. If your team is not full you cannot participate in a match.

iii. If all of a player's athletes are injured they can no longer compete.

iv. If all of a player's athletes are injured during a match they lose the match and do not receive any money or points

v. If a player wins a match (by having the highest score) they are rewarded with money and points

4. Playing a Match

   (a) Within a match athletes should compete in some meaningful way for example:

      i. Each team is organised into positions and the athletes face off against the athlete at the same position in the opposing team. Athletes in a defensive position will use their defence stat, while athletes in an offensive position will use their offence stat. The athlete with the higher stat will increase the total score of their team. The team with the highest score wins.

      ii. When the match finishes all athletes will lose stamina. If an athlete loses their face off their stamina should be reduced further.

      iii. If an athlete's stamina reaches 0, they are considered injured and can no longer compete.

   (b) Once a match has concluded the outcome should be clear, and the updated status of the team should be shown to the player.

   (c) You may optionally show each step in a match, with the current athlete match up and their relevant stats.

5. Visit the market and:

   (a) While at the market, the player must be able to see their current money.

   (b) Athletes and items can be drafted back to the market.

   (c) View athletes that are available for contracting including their price and their stats

      i. There should be 3 to 5 athletes to choose from

      ii. When an athlete is contracted the player will have the option to add them to their team or as a reserve.

      iii. If an athlete is added to the team, an existing team member should become a reserve and the new athlete will take their place.

      iv. The type of athletes sold may depend on other factors (e.g., a rarity system could be used, with rarer athletes becoming more common [or being unlocked] in later weeks).

   (d) View items that are for sale including their price and effect

      i. There should be at least three items in the market

      ii. When an item is bought, it should go to the player's inventory

iii. Items should enhance athlete stats

6. Take a bye (move to the next week)
   (a) All items/athletes in the market are updated (possibly randomly)
   (b) All matches are updated (following 3)
   (c) Each athlete restores their stamina back to full
   (d) There should be an option to specially train one athlete which will significantly improve their stats

There will also be some random events you will need to implement. These will only happen while your team is resting. The chances of these events occurring may depend on the difficulty setting. The player should be alerted when any of these random events occur:

1. An athlete's stat is increased
   (a) There should be a small chance an athlete gets a stat boost while resting (you may wish to tie this into other mechanics, such as increasing with the number of face offs won in matches, or whether or not the athlete was injured that week)
2. An athlete quits
   (a) The chance should increase if the athlete was injured the previous week(s)
   (b) The chance should be quite low
3. **A random new athlete joins**
   (a) The chance should increase depending on how many free slots the player has in their reserves.
   (b) The chance should be quite low

2.3 Finishing the Game

The game ends when one of the following occurs:

1. All weeks have passed.
2. The player does not have a full team nor enough money to buy more athletes.

Upon ending a screen will display the team's name, the selected season duration in weeks, the amount of money gained, and the amount of points gained.

2.4 Extra Credit Tasks

If you have finished early, and have a good looking application, then you will be in for a very high mark. If you want some more things to do, then please discuss it with the tutors in the lab, but you are free to add any features you wish, just be careful to not break anything. Here are some ideas, and you do **NOT** need to implement them all to get full marks:

- You may want to give athletes specific roles; each having different bonuses such as certain stats being increased/decreased, an advantage if they are in a specific position of the team, or a whole team bonus if the roles are correctly spread.
- You may want to allow the player to enter a 'seed' during setup that determines how random events of the game will play out.
- You may want to show the matches with sprites or animations.
- You may want to allow the player to save the current state of the game, and be able to load it up at a later time.
- You may want to add a story line to your game, including consistent characters, and a plot which gets told through pop ups or dialogue.
- You may want to add rarity levels for athletes which are properly tied in via some mechanic, such as increasing their chance to appear/unlocking them in the shop after some milestone (e.g. number of weeks or number of matches won).
- You may want to add special characteristics to athletes to make them more unique. For example, an athlete may only restore stamina every two weeks, but start with higher base stats, or an athlete may use their highest stat regardless of position.
- You may want to add some artwork.
- You may want to include some music or sound effects. We recommend you use resources such as pixabay.com or freesound.org for free to use images and sound. It is your responsibility to adhere to any licensing terms and they should be referenced in your README.

3 Design and Architecture

This section provides some **ideas** on how your program should be structured. The following are some class candidates, and should be represented in your program. Important things to think about here are interactions between classes, what classes should be instantiated, how you could use interfaces and/or inheritance, and how you could apply the design patterns you've learnt and will learn in SENG201.

3.1 Athlete

There shall be different athletes. Four shall be enough. Each athlete has at least a name, stamina, offence stat, defence stat and current health.

- An *optional* suggestion is to have generic roles for athletes (e.g., Attacker, Defender, All-Rounder), from which each named athlete inherits. Athletes can then be classified into these roles to allow for more diverse gameplay as discussed in section 2.4 Extra Credit Tasks

3.2 Item

There shall be different types of items (e.g., training equipment and foods). Four shall be enough. Each item should increase one or more of an athlete's stats when consumed.

3.3 Purchasable

Both athletes and items can be bought from a market. These both require a contract price, a sell-back price, and a description.

3.4 Match

Matches shall be generated based on some game factor (e.g., current week or difficulty) with some randomness. A match may be with another 'team', in this case this other 'team' must have a name. A match consists of a number of athletes that compete against the player's current team.

3.5 Random Event

A random event represents a circumstance that can happen while the team is resting. It should include behaviour to handle the event.

3.6 Game Environment

The game environment contains your game, and will implement functions to provide the options mentioned above, and will call methods of the above classes to make those options happen. The game environment keeps track of the game and should handle requests from the user interface and report back updates.

Try to keep your code modular and avoid putting user interface (UI) code in your game environment class. Keeping the game logic separate from the UI code will ensure that you can create both a command line interface and a GUI interface without having to alter your game environment.

4 Assignment Tasks

4.1 Sketching UML

Before you start writing code, sketch out a UML use case diagram detailing the program's users (actors) and their interactions with the system (use cases). This diagram will help you to identify the main functionalities of the program and to visualize its scope.

In addition to this diagram, create a UML class diagram of how you think your program will look like. It will help you get an idea of what classes there are, what class attributes are required, and will get you thinking of what classes communicate with other classes (call methods of another class).

4.2 Implementing a Command Line Application

Begin implementing classes, starting with athlete, team, store, items, and the game environment. Make a command line UI class that drives a simple command line application that works in a simple runtime loop which:

1. Prints out a list of options the player may choose, with numbers next to the options.
2. Prompt the player to enter a number to complete an option.
3. Read the number, parse it and select the correct option.
4. Call the method relating to the option and if necessary:
   (a) Print out any information for that option, such as select a battle.
   (b) Read in the number for the information offered.
   (c) Parse the number and complete the action.
5. Go back to step 1.

This will enable you to slowly build up features, and we recommend to only implement one feature at a time. Make sure you test your feature before moving onto implementing more features. Once you are feature complete and have a working game, you may move onto implementing a graphical application.

The command line application will only be assessed if there is no graphical application, or if there are fatal bugs in the graphical application. In this case, a penalty will be applied (see Section 6.4). **Hint:** For a very basic solution to read input from the command line you may want to explore class Scanner in the Java API.

4.3 Implementing a Graphical Application

You will be implementing a graphical application for your game using **Swing**, which will be explained in labs. For the purposes of this assignment, we do not recommend writing the Swing code by hand, and instead using the interface builder offered by the Eclipse IDE. This lets you build graphical interfaces in Swing by dragging and dropping components onto a canvas onscreen, and it will automatically generate the code to create the graphical application you built.

Please note, you are required to ensure that any automatically generated code complies with the rest of your code style, so you will need to change variable/method names and code layout.

Once you have built your interface, the next task is to wire up the graphical components to the methods your command line application supplies, and to update the onscreen text fields with the new values of your class attributes/member variables. Most of these functions are triggered on `onClick()` methods from buttons. Start small, and complete Section 2.1 "Setting up the Game" first to get used to GUI programming. You might need to slightly adjust your methods to achieve this. Then move onto the main game.

Note that this is the largest task to complete and many students under estimate how much time it will take. Try to be at this stage one week after the term break if possible.

4.4 Writing Javadoc

Throughout your application, you need to be documenting what you implement. Each attribute of a class should have Javadoc explaining what its purpose is. Each method needs to explain what it does, what variables it takes as parameters, what types those variables are, and what the method returns (provided it does not return void). You should be building your Javadoc regularly, as it integrates into the IDE very nicely, and will aid you in writing good code.

4.5 Writing Unit Tests

You should design JUnit tests for your smaller, basic classes, such as Athlete, Team, Store, and their descendants if you think necessary. Try and design useful tests, not just ones that mindlessly verify that getters and setters are working as intended.

4.6 Report

Write a short two page report describing your work. Include on the first page:

- Student names and ID numbers.
- The structure of your application and any design decisions you had to make. We are particularity interested in communication between classes and how interfaces and/or inheritance were used. You might want to reference your UML class diagram.

- An explanation of unit test coverage, and why you managed to get a high/low percentage coverage.

Include on the second page:

- Your thoughts and feedback on the project.
- A brief retrospective of what went well, what did not go well, and what improvements you could make for your next project.
- The effort spent (in hours) in the project per student.
- A statement of agreed % contribution from both partners.

4.7 A note on effort distribution

The "typical" or "common" distribution of the overall effort spent on these activities is: around 5% creating the UML diagrams; 20% developing the command line application; development of the graphical application is the most time consuming task taking around 50% of your time; documenting your code would take 5%; creating the unit tests around 15% and writing the report would take the last 5%.

However, note that these numbers vary between students and these percentages are not supposed to be the exact amount of effort invested in each task. They are only a rough guideline (e.g. we would not expect you to spend 50% of your time on creating UML diagrams or writing the report; on the other hand, we would expect that implementing the graphical user interface takes quite a substantial portion of the effort).

5 Deliverables

5.1 Submission

Please create a ZIP archive with the following:

- Your source code (including unit tests). We want your exported project as well so that we can easily import it into Eclipse.
- Javadoc (already compiled and ready to view).
- UML use case and class diagrams as a PDF or PNG (do not submit Umbrello or Dia files; these will not be marked).
- Your report as a PDF file (do not submit MS Word or LibreOffice documents; these will not be marked).
- A README.txt or README.md file describing how to build your source code, import your project into Eclipse and run your program.

- A packaged version of your program as a JAR. We must be able to run your program **in one of the lab machines** along the lines of: `java -jar usercode1_usercode2_SportsTournament.jar`.

Submit your ZIP archive to LEARN before the due date mentioned in Section 6.3. **Only one member** of the team is required to submit the ZIP archive.

5.2 Demos

During the last week of term, you will be asked to demo your project during lab and lecture time. Each team member must be prepared to talk about any aspect of your application, we will be asking questions about any and all functionality. There will be a form on LEARN in which you can book a time slot. Ensure you are both available, as you must attend the demo as a pair.

Only one member of the team needs to book a slot for the team, but both members are expected to attend the demo at that time; do not double book slots.

6 Marking scheme

6.1 Overall Assignment [100 marks]

6.1.1 Functional suitability and Usability [45 marks]

We will be testing the extent to which your code meets the requirements using the graphical interface. This includes running the program and executing its main functionalities.

If your graphical application is broken or faulty, partial marks will be awarded for your command line application.

6.1.2 Code quality and Design [20 marks]

We will be examining the code quality and design of your program. This includes: your naming conventions, layout and architecture, and use of object oriented features, among others.

Quality of your comments is also very important. You may wish to run `checkstyle` over your code before you hand it in.

6.1.3 Documentation [10 marks]

We will be looking at your use of Javadoc, and how well it describes the attribute or method you are commenting on, and how well it covers your codebase.

6.1.4 Testability [15 marks]

We will run your unit tests to see how well they cover your code, and we will examine the quality of those tests. Try to make your tests do something other than verifying that getters and setters work.

6.1.5 Report and UML diagrams [10 marks]

Your report and UML diagrams will be marked based on how well written the report is and the information the diagrams convey about your program. The report must include what is specified in section 4.6.

Employers place a lot of value on written communication skills, and your ability to reflect on a project, especially in agile programming environments.

6.2 Project Demos

For your project demo, you will have to log in to a lab machine, download your project submission from LEARN and import it into Eclipse. From there, you and your partner will be showing the examiners how your program works and you may be asked to:

- Construct a new game, and draft a team.
- View your inventory.
- View the properties of your team.
- Choose a match and play.
- Visit the market and view, buy and/or sell items.
- Show that random events occur.
- Show that the game can be completed.
- Show that the game runs without errors, obvious bugs or crashes.
- Fulfils any or all of the requirements set.
- You may be asked to explain how your graphical interface is written, and point to specific code.
- You may be asked about how interfaces and/or inheritance work in your program, again pointing to specific code.
- Anything else that the examiner wishes to ask about.

If you do not turn up to demo in your time slot, you will be penalised (see Section 6.4). The project demo instructions may change depending on multiple factors (e.g., COVID-19 restrictions), in case of any changes, this will be informed through LEARN.

6.3 Important Dates

- **Project launch:** March 28, 2023.
- **Weekly progress deadlines (to be recorded via LEARN):**
  - **Week 6** (28 March - 3 April, 2023) progress must be recorded by April 5, 2023 at 11:59pm.
  - **Week Break-1** (4-10 April, 2023) progress (if any) should be recorded by April 12, 2023 at 11:59pm.
  - **Week Break-2** (11-17 April, 2023) progress (if any) should be recorded by April 19, 2023 at 11:59pm.
  - **Week Break-3** (18-24 April, 2023) progress (if any) should be recorded by April 26, 2023 at 11:59pm.
  - **Week 7** (25 April - 1 May, 2023) progress must be recorded by May 3, 2023 at 11:59pm.
  - **Week 8** (2-8 May, 2023) progress must be recorded by May 10, 2023 at 11:59pm.
  - **Week 9** (9-15 May, 2023) progress must be recorded by May 17, 2023 at 11:59pm.
  - **Week 10** (16-22 May, 2023) progress must be recorded by May 24, 2023 at 11:59pm.
  - **Week 11** (23-23 May, 2023) progress must be recorded by May 25, 2023 at 11:59pm.
- **Last day for registering teams:** April 6, 2023 at 5pm. After this time, the teaching team will randomly allocate those students without a pair. Allocations will be final.
- **Last day for booking a time slot for the demo:** May 24, 2023 at 5pm. After this time, the teaching team will allocate time slots to those teams without a booked slot. Allocations will be final.
- **Project submission due date:** May 23, 2023 at 5pm.
- **Lab demos:** during lab and lecture time of 29 May - 2 June, 2023.

6.4 Penalties

- **-6 marks** for failure to register your team by the given deadline.
- **-4 marks** for failure to record your weekly progress by the weekly deadline, see Section 6.3 for details. This penalty applies as soon as the project starts and regardless of whether or not you have a team. The penalty will be applied only to the student in the pair who failed to record their weekly progress.

The penalty applies as many times as you fail to record progress. For example, if you missed it twice, a penalty of -8 marks is applied (2 x (-4) = -8).

It is not allowed to record the weekly progress in one go at the end of semester.

Students who record their weekly progress in a timely manner for all weeks (excluding the break weeks) will be granted a one-time bonus of +5 marks. This will only apply to the student in the pair who recorded their weekly progress not later than two days after the reported week ended.

You are not expected to work on the project during the term break. However, if you decide to do this, it would be beneficial for you to keep recording your weekly progress.

- **-6 marks** for failing to book a time slot for the demo by the given deadline, see Section 6.3 for details. It applies to both members of the team. This penalty also applies if both members of the team book two time slots.
- **-15 marks** for submitting after the due date PLUS **-1 mark** per each hour of delay. It applies to both members of the team and it is capped up to -100 marks, i.e., the total marks of the assignment.

  For example, if you submit 16 hours* after the due date, a penalty of -31 marks will be applied [(-15) + (-16) = -31].

  * For the purposes of this calculation, the ceiling function is used.
- **-30 marks** for not providing a GUI for your application.
- **-30 marks** for failing to show up to the demo. It applies only to the student who missed the demo.
- **-100 marks** if plagiarism is detected.

7 FAQ

I cannot see my group number, can you please let me know either what it is or how to figure it out?

Your group number is the three-digit number in the group description.

I can't find the Participants section, could I be pointed in the right direction?

Open the normal SENG201 LEARN page, then click the tab labelled 'Participants' under the course header. If you still can't find it, use CTRL+F and search "Participants"

I submitted my project three seconds past the deadline, is it counting as a late submission?

Yes, please refer to the subsection 6.4.

We were just submitting the project while we were disconnected from the WiFi. This made our submission 15 seconds late. Is there any chance we can not be marked down for this?

No, please refer to the subsection 6.4.

I forgot to pick a time slot for my demo, could you reopen the demonstrating slot chooser?

No, please refer to the subsection 6.4. Your demo slot will be allocated by the teaching team.

Our ZIP file containing our report, Javadoc, UML class diagram and a .JAR file was submitted, however, we forgot to add the source code. Is our submission incomplete?

Yes, the submission is incomplete. Not submitting your source code will void your submission (i.e., you will only be able to get the marks associated to the Report, see subsection 6.1).

We are currently having difficulties with LEARN. Is there any chance we could ask for a small extension of around 15 minutes to fix this issue?

No, please refer to subsection 6.4.

We realized that we forgot to add the project report, so we had to upload our project again. This resulted in our submission missing the deadline by a few seconds. Is it possible to get the 15-mark penalty removed?

No, please refer to subsection 6.4.

We are thinking of deleting the CLI code from the final deliverable as the GUI runs fine. It's [the CLI] not being used at all anymore. Is this fine?

Yes, it is OK to remove the CLI code if your GUI is working well. However, only the files you submit will be marked. In case your GUI fails, you won't be able to use your CLI project.

My group partner has booked a time slot, do I need to book in as well?

No, only one member of the team is required to book a time slot.

We're getting pretty close to finishing our project, and we'd like to maybe put in some background music to go with our GUI application.

What would we have to do with respect to copyright if we wanted to include this?

The safest path to follow is not using copyrighted material. We suggest finding music under royalty-free or creative commons licenses.

Is it necessary for the project to be able to be imported into Eclipse so that the markers can build it, or can we specify in our README that it is an IntelliJ project?

Yes, it is required to be able to import your project into Eclipse. The README may include instructions for IntelliJ but must include the Eclipse ones too.

How many marks is it possible to achieve for just submitting a command line application of the game?

You would be losing marks (see subsection 6.4), but the overall marks also depend on the rest of the application, how much functionality you implemented, the quality of your code, testing practices, etc.

Does our command-line app need to be included in the final class diagram?

No, the CLI app (and its related documentation, if any) will not be marked unless the GUI does not work.

If your getters and setters don't contain any logic (i.e., they are trivial) then there is no need to test them thoroughly.

You can test this type of getters/setters (those without any logic) in conjunction with more complex methods.

About the class diagram, do we need to include the launch screen and close screen methods in the class diagram.

There is no need to include those methods (trivial getters, setters, launch, close) in the class diagram.

I am wondering whether or not my SENG201 project partner and I must submit a report each, or one report between us.

You must submit one report per team.

I'm wondering as to the degree to which we can reuse code from the labs/tutorials in our own project.

If you use code from the labs or tutorials you should mention this in the code comments and the report.

Given that testing is an important part of the project, I was wondering how we are supposed to approach testing for the GUI with Java Swing.

There are several options for testing GUIs automatically (e.g., Katalon Studio), however, these type of tests are out of the scope of SENG201. You are required to run manual tests for testing your GUI. Manual testing requires you to prepare a plan (e.g., a list of key functionalities) of what you will test and how you will do it. For example, to test that the scenario "once the user has bought a player, it will be shown in their inventory" you will actually need to play the game and define a set of steps to test this scenario (Click on "Go to the Market" button; Select a player; Click on the "Buy" button; Go to my inventory, Check that the player I bought is there). As you can see, testing the GUI requires more complex test cases than unit testing.

For the manual testing, what I am getting is that we need like a script/plan on what to do with the GUI and what is expected to happen. Do we need to document this script/plan and submit it as well because they are technically part of the test?

Ideally, you should document the manual test plan to keep a consistent track of your tests. It is very likely that you will need to execute the tests more than once, so having a well-documented plan is desirable. However, it is not part of the deliverables required for the SENG201 project.

We have made the class diagram for the model classes a while back, but we haven't included the view. When I was thinking about what the diagram for the view would look like, I realized that a lot of them if not all of them will just be very simple diagrams pointing back to the main class from the model. Therefore, I am not sure whether it's really necessary.

Diagrams should be understandable, so there is no need to add relationships or classes that are not adding anything new/relevant to them. A note in your report explaining how your application behaves in addition to an up-to-date class diagram would be enough for others to understand your application.

Just wondering what the best way of testing randomness in the project would be? Are we able to assume that the Java Random library works to do this and just perform manual testing to ensure it runs correctly, or is there a way to do this with JUnit?

Testing randomness can indeed be challenging, and finding a specific frequency of an action even more so. There are a few ways to go about testing randomness: Expose the random implementation so that you can specify a seed you know will reproduce the expected result. You will have to find a seed for Random that causes the action you are expecting, then it should happen every time without fail. You can create functions that produce the specified action based

on an input, which in your program will be generated from a random. But when testing you can hard-code the value coming in so that it that causes the expected action. E.g. if we have a 50/50 chance where any even number triggers the action, we can simply pass in an even number during testing and pass in the value from random.nextInt() during execution. You can repeat a random action many times and assert that it happens at least once. For example if you have a 50/50 chance for something to happen, you could run it 10 times which based on probability will pass 99.9% of the time. Of course having tests that aren't completely deterministic isn't great. But with 99.9% probability generally it will work enough that it is negligible. To find frequency, we can use the same idea as above, in this case running a larger number of tests. And finding the distribution of each result you are testing, then match this to what you are expecting. I recommend not being strict with this, e.g. if it is supposed to happen 50% of the time don't check for exactly this, instead check for boundaries close to this. The size of these boundaries will depend on how many tests are ran overall, and what success rate you are targeting (I would suggest around 99.9% probability of tests passing, as when more tests that have a chance to fail are added, the overall likelihood of all tests passing is reduced). Overall, I would suggest staying away from non-deterministic tests where possible. For example if you have a function that takes an integer input (see bullet point 2) between 0-100 and any value below 10 produces an action, simply test that this indeed happens for some value less than 10 and doesn't happen for values 10 and above. Then you can simply assume that the Random.nextInt(100) will provide a random number within the range as expected.

We are planning to use images in our game and wonder if we can use images from internet (Google, etc.)?

You can use images from the internet if and only if you are authorized to use them. The safest path to follow is using non-copyrighted material. We suggest finding images under royalty-free or creative commons licenses.

Are we allowed to use ChatGPT or similar tools?

In this case, we discourage the use of tools like ChatGPT. However, if, for whatever reason, you use ChatGPT or similar tools you are fully responsible for any copyright or license violation as well as any bugs, design problems and other issues introduced to your code. Furthermore, tutors will not provide help with any automatically generated code. Also, you must be able to fully explain any code that you take from sources other than your own writing. Finally, **you must provide the chat as part of your deliverables (failing to do so will be considered plagiarism)**. Be aware that you must able to demonstrate a full understanding of what you delivered (e.g., software design and implementation decisions).