# Project

## Aims

- Appreciate issues in user interface design
- Learn practical aspects of graphical user interface programming
- Learn more about the Java class libraries
- Learn the application of design patterns.

## Due Dates

Milestone 1: 5PM Sunday Week 5 (Feedback: Week 6 Lab)

Milestone 2: 5PM Sunday Week 7 (Feedback: Week 8 Lab)

Final milestone: 5PM Sunday Week 9 (Demonstration: Week 10 Lab)

**NOTE:** There is **NO** provision for late submissions with a late penalty. Not submitting by the deadline is considered non-submission.

## Value: 25 marks

---

## Overview

You have received a request from a client for an application for the playing of dungeon-style puzzles. With a partner from your lab class, you will follow an agile development process to design and implement a desktop Java application that satisfies the requirements of the client (see below). The final piece of software you deliver is expected to be of professional quality, user-friendly, and demonstrate the knowledge and skills you have acquired in this course.

## Partner

Your first step in this project will be to find a partner. You will do this under the guidance of your tutor in the week 3 lab. You may work as an individual if you wish, but your project will be judged by the same criteria as if you had done it with a partner. If class sizes necessitate, groups of 3 will be allowed.

After resolving who you will be working with, you should follow the instructions in the week 3 lab for registering your group. A repository on GitLab will be created for you at the end of week 3 based on the group you create, so you **must** create one by then.

Once created, your repository will be available here (replace*GROUP_NAME* with your group's name):

https://gitlab.cse.unsw.edu.au/COMP2511/19T3/GROUP_NAME

## Project setup

**NOTE**: For the first milestone, it is not necessary to set up the project in Eclipse.

Because this project uses JavaFX, it requires some additional setup steps in Eclipse. It's relatively straightforward if you're using a CSE computer, but if you're using your own computer you will need to download an alternate JDK.

1. **(Only necessary for non-CSE computers)** Go here and download the Java 11 Zulu JDK FX for your OS. Unzip it once it finishes downloading.
2. In Eclipse, import this project as normal. You should see an exclamation mark on the project indicating an error.
3. Go to **Window -> Preferences**. On the left, under **Java** select **Installed JREs** then click **Add**. Ensure **Standard VM** is selected and click **Next**.
4. 
   - **(On CSE computers)** Enter exactly `/home/cs2511/jdk-jfx` into **JRE Home**. The other fields should fill in automatically.
   - **(On non-CSE computers)** Click **Directory** and select the directory of the Zulu JDK you unzipped in step 1. Change **JRE name** to exactly `jdk-jfx`.
5. Click **Finish** then **Apply and Close**

If these steps worked, the project should no longer have the exclamation mark on it and you should be able to run the starter code.

## Preliminary client requirements

The client desires an application that lets the user move a player around a dungeon and try to overcome various challenges in order to "complete" the dungeon by reaching some goal. The simplest form of such a puzzle is a maze, where the player must find their way from the starting point to the exit.

Maze

More advanced puzzles may contain things like boulders that need to be pushed onto floor switches,

Boulders

enemies that need to be fought with weapons, or collectables like potions and treasure.

Advanced dungeon

### Dungeon layout

To be specific, the layout of each dungeon is defined by a grid of squares, each of which may contain one or more entities. The different types of entities are as follows:

| Entity | Example | Description |
|---|---|---|

| Entity | Example | Description |
|---|---|---|
| Player | Player | Can be moved up, down, left, and right into adjacent squares, provided another entity doesn't stop them (e.g. a wall). |
| Wall | Wall | Blocks the movement of the player, enemies and boulders. |
| Exit | Exit | If the player goes through it the puzzle is complete. |
| Treasure | Treasure | Can be collected by the player. |
| Door | Door Door | Exists in conjunction with a single key that can open it. If the player holds the key, they can open the door by moving through it. Once open it remains so. The client will be satisfied if dungeons can be made with up to 3 doors. |
| Key | Key | Can be picked up by the player when they move into the square containing it. The player can carry only one key at a time, and only one door has a lock that fits the key. It disappears once it is used to open its corresponding door. |
| Boulder | Boulder | Acts like a wall in most cases. The only difference being that it can be pushed by the player into adjacent squares. The player is only strong enough to push **one** boulder at a time. |
| Floor switch | Floor switch | Switches behave like empty squares, so other entities can appear on top of them. When a boulder is pushed onto a floor switch, it is triggered. Pushing a boulder off the floor switch untriggers it. |
| Portal | Portal | Teleports entities to a corresponding portal. |
| Enemy | Enemy | Constantly moves toward the player, stopping if it cannot move any closer. The player dies upon collision with an enemy. |
| Sword | Sword | This can be picked up the player and used to kill enemies. Only one sword can be carried at once. Each sword is only capable of 5 hits and disappears after that. One hit of the sword is sufficient to destroy any enemy. |
| Invincibility potion | Invincibility | If the player picks this up they become invincible to enemies. Colliding with an enemy should result in their immediate destruction. Because of this, all enemies will run away from the player when they are invincible. The effect of the potion only lasts a limited time. |

## Goals

In addition to its layout, each dungeon also has a goal that defines what must be achieved by the player for the dungeon to be considered complete. Basic goals are:

- Getting to an exit.
- Destroying all enemies.
- Having a boulder on all floor switches.
- Collecting all treasure.

More complex goals can be built by logically composing basic goals. For example,

- Destroying all enemies AND getting to an exit

- Collecting all treasure OR having a boulder on all floor switches
- Getting to an exit AND (destroying all enemies OR collecting all treasure)

If getting to an exit is one of a conjunction of conditions, it must be done last. For example, if the condition is to destroy all enemies AND get to an exit, the player must destroy the enemies *then* get to the exit.

## Input

Your application will read from a JSON file containing a complete specification of the dungeon (the initial position of entities, goal, etc.). Example dungeons are included in the `dungeons` directory and the starter code contains an incomplete dungeon loader.

The dungeon files have the following format:

> { "width": *width in squares*, "height": *height in squares*, "entities": *list of entities*, "goal-condition": *goal condition* }

Each entity in the list of entities is structured as:

> { "type": *type*, "x": *x-position*, "y": *y-position* }

where *type* is one of

> ["player", "wall", "exit", "treasure", "door", "key", "boulder", "switch", "portal", "enemy", "sword", "invincibility"]

The `door`, `key`, and `portal` entities include an additional field `id` containing a number. Keys open the door with the same `id` (e.g. the key with `id` 0 opens the door with `id` 0). Portals will teleport entities to the**one** other portal with the same ID.

The goal condition is a JSON object representing the logical statement that defines the goal. Basic goals are:

> { "goal": *goal* }

where *goal* is one of

> ["exit", "enemies", "boulders", "treasure"]

In the case of a more complex goal, *goal* is the logical operator and the additional *subgoals* field is a JSON array containing subgoals, which themselves are goal conditions. For example,

```
{ "goal": "AND", "subgoals":
  [ { "goal": "exit" },
    { "goal": "OR", "subgoals":
      [ {"goal": "enemies" },
        {"goal": "treasure" }
      ]
    }
  ]
}
```

Note that the same basic goal *can* appear more than once in a statement.

You can extend this format to include additional information if you wish, but your application should still work with files in the original format.

### User interface

The UI component of this project will be implemented in JavaFX. The starter code contains a very basic UI showing how a player can be moved around with the arrow keys, but it is missing many features (the player can walk through walls for one).

The client has given you free reign over the visual design of the program. Included in the starter code are some example assets, but you are free to use different ones. You can find them elsewhere or even create your own. The examples above came from here.

## Requirement analysis (Milestone 1)

For this initial milestone, you are to model the requirements of the client as user stories on the issue board in GitLab. You will show these user stories to your tutor in the Week 6 lab, where they will give feedback.

It's important that you an your partner meet and collaborate early in order to develop a shared understanding of the project and how you intend to approach it. In developing the stories you will need to consider the requirements as given by the client in this document, but also make your own judgements about the expectations of potential users. Epic stories should be broken down into user stories and each story should have its own card.

The default columns that GitLab provides are sufficient for this project.

You are expected to produce:

1. High-level epic stories from the problem statement. Each epic should have its own card/issue and a corresponding tag used to mark user stories that fall under it.
2. User stories, each containing:
    - a short description of the feature based on the Role-Goal-Benefit (or Role-Feature-Reason) template (Refer to the RGB model from COMP1531 if unsure)
    - an estimate for the implementation of the user story in user story points (e.g. 4 points).
    - a tag indicating the priority
    - acceptance criteria for each user story as a checklist in the issue (Refer to material from COMP1531 if unsure)

As you progress through the rest of the project, you will keep your board and issues up to date: checking off acceptance criteria that have been satisfied and moving stories from **To Do** into **Doing** and finally into **Closed**.

## Domain modelling and backend implementation (Milestone 2)

Based on your requirements analysis, and taking into account the feedback from your tutor, you will produce a domain model for the backend component of your project in the form of a conceptual UML class diagram, implement it in Java and write JUnit tests to test its functionality.

In deciding on your design and writing your implementation, you should follow the practices and design principles covered in the course. You are expected to apply at least 3 of the design patterns covered in the course. It is up to you where they are applied, but you will be expected to justify how and why you used them to your tutor during demonstration.

Your class diagram only needs to be conceptual (showing the general structure of the classes and their relationship), but it needs to be consistent with the code and clearly indicate where you're using design patterns (use labels if necessary).

Your JUnit tests should be rigorous to ensure your backend functions as expected. In addition to basic unit tests, you need to have tests based on your acceptance criteria.

In the week 8 lab, your tutor will ask you questions about your design and implementation and give feedback you can use as you continue to work on it.

## UI design and extensions (Milestone 3)

For this milestone you are to design and implement the user interface component of the application. A very basic UI can be built with minimal changes to the starter code, so that is where you should start. Fancier UI features can be added once you have something that is at least usable. You should apply the ideas from user-centric design and consider the usability heuristics covered in the lectures.

Additionally, for this milestone, you also have the chance to extend the project with your own ideas. Note that, to get high marks for these extensions, you will need to consider how they impact the user. **Extensions that are technically complex, but do not provide the user with any real benefit are not considered good extensions**. You can, and should, create additional user stories to model the requirements of these extensions. Possible extensions include but are not limited to, multiplayer, different sorts of enemies, new weapons, and animated movement.

This final milestone will be a culmination of all the work done in the previous milestones. You have the opportunity to improve on your design based on feedback from your tutor. Marking of the design will be harsher for the final milestone as you have already had the opportunity to receive feedback.

## Assessment

You will be assessed on your ability to apply what you have learnt in this course as well as your ability to produce a significant piece of software.

In cases where the client has not been explicit in their requirements, you will need to make your own design decisions with your partner. However, this does not mean you can ignore whatever requirements the client has given you. You may be asked to justify any assumptions you have made during marking.

You are expected to use git appropriately by committing regularly with meaningful commit messages and using feature branches to manage significant changes. Similarly, you should use the task board to coordinate work with your partner. You will need to take the principles you learnt from COMP1531 and apply them here.

While it is up to you how to divide the work between you and your partner, both of you are expected to contribute code. Just creating diagrams and documentation is not sufficient contribution. Both members of a pair *generally* receive the same mark, but if there is a significant imbalance in the amount of work done, the total mark may be scaled to match actual contribution.

## Hints

- The first two milestones do not require a working UI. First determine how you are going to model a dungeon and its entities before considering the UI. A well designed back-end will require minimal change to connect to the UI.
- Up until the JavaFX has been covered, you may find some of the starter code to be hard to understand. For the backend, you can either ignore the starter code completely, or just look at the `Dungeon`, `Entity`, `Player` and `Wall` classes.
- The starter code uses the observer pattern to ensure the frontend and backend are in a consistent state and that they are not tightly coupled. It would be advisable to do the same for the changes you will make.
- The majority of marks available (see below) are for having a well designed application that meets the requirements. Avoid adding extra complexity and extensions till you have something that meets the most basic requirements.

## Submission

### Milestone 1

You should have all your user stories entered into the issue board on your GitLab repository. You may continue to use the board between the deadline and your tutor's assessment, but they will be looking at the dates issues were modified to make sure you did the work that was required of you prior to the deadline.

### Milestone 2

Submit the contents of your GitLab repository with the following command:

```
$ 2511 submit milestone2
```

Your UML class diagram should be a PDF file at the root of your repository named `design.pdf`.

### Milestone 3

Submit the contents of your GitLab repository with the following command:

```
$ 2511 submit milestone3
```

You will demonstrate your application to your tutor in Week 10. You may be asked to justify your design decisions and explain how you worked with your partner.

## Marking criteria

The marks are allocated as follows:

- Milestone 1 (5 marks)
- Milestone 2 (8 marks)
- Final milestone (12 marks)

Below is a *rough* guide on how you will be assessed for each milestone.

### Milestone 1

| Criteria | Mark | |
|---|---|---|
| Stories | 0 | No user stories |
| | 1 | User and epic stories not in a valid format and/or vague or ambiguous |
| | 2 | User and epic stories in a valid format, but with unclear benefits, goals or acceptance criteria |
| | 3 | Unambiguous and clear user stories with concrete acceptance criteria |
| Planning | 0 | No user stories have points or priorities |
| | 1 | Only some user stories have points or priorities |
| | 2 | User stories have appropriate story point values and priorities |

### Milestone 2

| Criteria | Mark | |
|---|---|---|
| Completeness | 0 | No or largely incomplete backend |
| | 1 | Backend implements some of the entities |
| | 2 | Backend implements most of the entities |
| | 3 | Backend implements all of the entities |
| Testing | 0 | No JUnit tests |
| | 1 | JUnit tests for behaviour of a few entities |
| | 2 | Rigorous JUnit tests for behaviour of all entities |
| Design | 0 | No apparent consideration for design |
| | 1 | Messy design and diagrams and/or design inconsistent with code |

| Criteria | Mark | |
|---|---|---|
| | 2 | Clear design and diagrams with partial adherence to design principles and patterns |
| | 3 | Clear design and diagrams fully adhering to design principles and patterns and conforming to code |

Marks will be deducted for poor git and GitLab usage. For example, meaningless commit messages, large commits, issue board out of date, etc.

### Milestone 3

| Criteria | Mark | |
|---|---|---|
| Completeness | 0 | No or largely incomplete project |
| | 1 | Dungeons can be played with most of the entities |
| | 2 | Dungeons can be played with all of the entities |
| Design | 0 | No apparent consideration for design |
| | 1 | Messy design and diagrams and/or design inconsistent with code |
| | 2 | Messy diagrams and/or poor application of design patterns |
| | 3 | Clear design and diagrams with partial adherence to design principles and patterns |
| | 4 | Clear design and diagrams fully adhering to design principles and conforming to code, and correct application of design patterns |
| Interaction | 0 | Very basic user interface |
| | 1 | Interface that makes it possible to solve dungeons, but is slow, awkward, or buggy |
| | 2 | An interface that is mostly usable but with little consideration for usability heuristics |
| | 3 | Interface that is easy to use |
| | 4 | A product that is engaging, intuitive and fun to use |
| Extensions | 0 | No extensions or only very basic extensions |
| | 1 | One extension that represents some technical consideration |
| | 2 | Multiple extensions that represent some technical as well as design and user interaction consideration |

Marks will be deducted for poor git and GitLab usage. For example, meaningless commit messages, large commits, issue board out of date, etc.