Project

Aims:

- Gain experience working in a small team-based environment
- Increase familiarity with problem-solving algorithms
- Appreciate issues in user interface design
- Learn practical aspects of graphical user interface programming
- Learn more about the Java class libraries

Due Dates: Week 12, Friday, May 25, 11:59 p.m. (project); Sunday, May 27, 11:55 p.m. (peer assessment and individual diary)

Value: 20%

Gridlock – Interactive Puzzle Game

Your task is to create an interactive puzzle game for the game of Gridlock (Rush Hour, Traffic Jam), where the objective is for the player to resolve the gridlock and move a designated car (usually the red car) through the exit and off the board and out of the traffic jam. The standard playing area is a 6x6 grid and there is only one exit, on the third row from the top at the extreme right. As cars (of size 2) and trucks (of size 3) can only move along their row or column, the red car must be positioned on the same row as the exit. Naturally, cars and trucks cannot move anywhere unless there are free squares for them to move on to. Variations on the standard game could be to have multiple players, a variety of levels, different sorts of obstacles, the necessity to collect items, limiting the number of available moves, etc.

Two games are shown below. The first is a very easy puzzle, while the second is very difficult. The objective is to move the red car through the exit, which is at the right hand side of the board on the same row (the exit is obscured by the yellow and purple trucks in these images). Cars and trucks can only move horizontally or vertically along a row or column, and of course, only the red car can exit the board.

In this assignment, the aim is to construct an *interactive* puzzle game that provides an interesting and challenging experience for the player. The primary requirement is not to develop an efficient problem solver (though that could be useful as part of the application), but to understand and anticipate user requirements, and to design a graphical user interface so that players of varying degrees of expertise can interact effectively with the system.

All programming should be done using standard Java. Do not use Java OpenGL or any third-party graphics library. However, you may use JavaFX under Oracle Java SE 8 rather than OpenJDK 8.

In this assignment, you will work in teams of **five students enrolled in your tut-lab** and follow the Scrum process. You will need to assign Scrum roles within the team and formulate an agile plan for your project (though all members should take on the developer role). A number of questions to help you determine the initial scope of the project are:

- who are the intended users: are they novice, intermediate or expert, or all sorts of users (do we even understand what this means?)
- what sort of features are basic to the application (i.e. needed by every user)?
- what sort of features are needed by different categories of users: how can the interface handle seemingly different requirements?
- what sort of help or hints (if any) should the system be able to provide to users (and how and when is this help given)?
- what platforms with what form factors is the system designed to run on?

Of course your scope will also include some basic "back-end" operations such as generating new puzzles, computing solutions, etc., as required.

You will be guided by your tutor throughout this project (think of your tutor as a representative client). You will be assessed individually on your contribution to software development, but also on the degree to which you follow good teamwork practice. To record your contribution, you should maintain an **individual diary** that contains your personal record of the project, answering, **for each day**, the three standard daily standup questions:

- what did I do yesterday?
- what will I do today?
- is anything in my way?

You will need to **show** (not e-mail) your diary to your tutor weekly (in Weeks 8, 9, 10, 11 and 12) to ensure your tutor is aware of your contributions to the project. Special arrangements will be made for Anzac Day. You will also need to set up and use a **git repository** for use in this project. Every team member should regularly commit work to the repository, as this forms a further record of your contribution to software development.

Your team will be organized as a Scrum team, and the project should be formulated as a number of user stories. You should use Trello as a **project management tool** to manage your user stories in a series of lists, to maintain the backlog and plan each sprint. Sprints can be either one or two weeks in duration. There are no set times for producing intermediate outcomes because your project plan is an agile plan of your own devising, and is subject to revision, however your tutor will advise you if your team seems to be falling behind the anticipated schedule in order to achieve a mimimum viable product. To help facilitate progress, the Week 10 tut-lab will be devoted to sprint reviews: each team will present their progress (no slides) and the current status of the plan; the rest of the class is expected to provide feedback. There are no marks for this activity, however feedback from the sprint review should be used to improve the quality of the system you eventually submit at the end of the project.

The team mark is for the final product. As a team, you will demonstrate your system to **two assessors** (your tutor and one other tutor, and possibly the lecturer in charge) in the tut-lab in Week 13. Your software will also be compiled and run on the CSE machines and/or other machines for further manual testing. Assessment will be based on effective interaction and user interface design, as well as correctness, generality and software design (see below for details). Your team presentation should **not** be a "sales pitch": your audience are all professional Software Engineers who are experts in object-oriented design and may ask you questions about your design (you are not presenting your software to the intended users of your system). There is no one "best" solution to this assignment. What is important is to produce a working system intended for some class of users and be able to justify your design decisions.

Your submission should include two Unix executable files, `compile` and `run`. When `compile` is called, your program should be compiled from the command line using `javac` commands. When `run` is called, your game should be executed using `java` commands. Make sure that all resource files (such as sounds and images) are loaded correctly, and that any files used by your program are saved locally and can be reloaded (for example, do not save such files in the user's home directory or in a parent directory).

It is an important part of this project for you (as a team) to work together and manage your own time. You need to follow the agile principle of making consistent and steady progress without reliance on overwork near the deadline, so you should start on time and not be overly optimistic about what you can achieve (remember you are all doing other courses with other assignments). It is essential for you as a team to monitor your progress and it is often necessary to revise the plan after a sprint review. The basic requirement is to develop a minimum viable product. Only consider extensions once the MVP is working satisfactorily.

Submission

- Submit your **project** in **one `.zip`** file per team using the following command:

```
give cs2511 project project.zip
```

- Your `.zip` file should contain:
    - All your **`.java` source files**, perhaps in a subdirectory (your Java files will be compiled on

various machines including under Java SE 8 on Mac OS 10.13.4)
    - Any **resources** (such as sound or image files) in an appropriate subdirectory (not necessarily where Eclipse stores them)
    - A `.pdf` file containing your team's **design documents** (a UML class diagram and, optionally, other diagrams necessary to understand your design)
    - The `.git` folder from your team's **git repository**
    - A **Unix executable file** called **`compile`** at the top level, which when called, compiles your Java program from the command line
    - A **Unix executable file** called **`run`** at the top level, which when called, executes your game from the command line
    - **At the "top level" means that when unzipped, the files should be in the same directory as the zip file, not in a subdirectory**
        - **Copy the project directory from Eclipse, place your `compile` and `run` scripts there, and zip the files in this directory**
        - **Check the contents of the zip file using `unzip -l project.zip` and look for `compile` and `run` at the top level**
        - **Refer to `sample-project.zip` for an example**

- Check that your submission has been received using the command:

```
2511 classrun -check project
```

- Submit your individual **peer assessment** via the Moodle Team Evaluation tool

- Submit your **individual diary** to your tutor via e-mail

Assessment

The team project mark comprises an individual component (5%) and a team component (15%) making up 20%. The individual component mark assesses contributions to software development and teamwork practices using as evidence a reflective diary shown weekly to the tutor, the git log for the project, the Trello task board and peer assessments of contributions; the team component assessment includes a demonstration of the final system held in Week 13. Each member of a team *generally* receives the same mark for the team component, however, where individual contributions to software development are highly unequal, individual marks for the team component will be adjusted to reflect the level of contribution. The Moodle Team Evaluation tool will be used to enable each team member to assess (anonymously) the relative contributions of the team members. These peer assessments, along with comments entered using the tool, will be taken into account in any mark adjustments.

Late penalty: There is no provision for late submission of the project. Late submissions receive a 0 mark or a mark at the discretion of the lecturer in charge. The same applies if the demonstrated system is found to differ from the submitted version.

Assessment Criteria

- Individual component: based on contributions to software development and teamwork practices as recorded in your **individual diary** and evidenced by the **git log** and the **Trello board**, and possibly adjusted based on **peer assessments**

- Team component: based on the final product as determined from your **demonstration** and **further**

testing, and possibly adjusted using **peer evaluations**. The scheme below is a guide only; note especially that marks for extensions will be reduced if the minimum viable product is not fully working:

- Correctness (3 marks)
  - 1: Serious bugs/system crashes during demo, or `compile` or `run` doesn't work
  - 2: Minor bugs/some strange behaviour, e.g. runs only on one platform
  - 3: No bugs seen during demo/testing
- Interaction (3 marks)
  - 1: Very basic user interface
  - 2: Easy to use interface with a range of features
  - 3: Smooth, responsive, intuitive, well designed user interface
- Design (3 marks)
  - 1: Messy design and diagrams and/or design inconsistent with code
  - 2: Clear design and diagrams with partial adherence to design principles
  - 3: Clear design and diagrams fully adhering to design principles and conforming to code
- Generality (3 marks)
  - 1: Predefined or simply generated puzzles
  - 2: Algorithm that generates reasonable puzzles
  - 3: Algorithm that generates interesting and challenging puzzles
- Extensions (3 marks)
  - 1: Basic game play with simple extensions
  - 2: Substantial extensions such as animation, multi-player, etc.
  - 3: More than one type of *significant* extension

Plagiarism

Remember that ALL work submitted for this assignment must be your own (team's) work and no code sharing or copying (between different teams) is allowed. You should **carefully** read the UNSW policy on academic integrity and plagiarism, noting, in particular, that *collusion* (working together on an assignment, or sharing parts of assignment solutions – between teams) is a form of plagiarism. You may use small snippets of code from textbooks or the Internet only with suitable attribution of the source in your program. **Do not under any circumstances submit large amounts of code obtained from the Internet.**