INFO1113 / COMP9003   Assignment

Task Description

In this assignment, you will create a game in the Java programming language using the Processing library for graphics and gradle as a dependency manager. In the game, players control tanks which can aim and fire at each other. Players gain score for hitting another player's tank, causing them to lose health. After all levels are completed, the player with the highest score wins.

You have been given the task of developing a prototype of the game. A full description of gameplay mechanics and entities can be found below. An artist has created a simple demonstration of the game and has posted it on your online forum (Ed). You can also play a similar game here.

You are encouraged to ask questions on Ed under the assignments category if you are unsure of the specification – but staff members will not be able to do any coding or debugging in this assignment for you. As with any assignment, make sure that your work is your own, and do not share your code or solutions with other students.

Working on your assignment

You have been given a scaffold which will help you get started with this assignment. You can download the scaffold onto your own computer and invoke gradle build to compile and resolve dependencies. You will be using the Processing library within your project to allow you to create a window and draw graphics. You can access the documentation from here.

Gameplay

The game contains a number of entities that will need to be implemented within your application.

Level

Each level is read from a text file of characters 28x20. The size of the window should be 864x640, meaning each character in the file corresponds to 32x32 pixels.
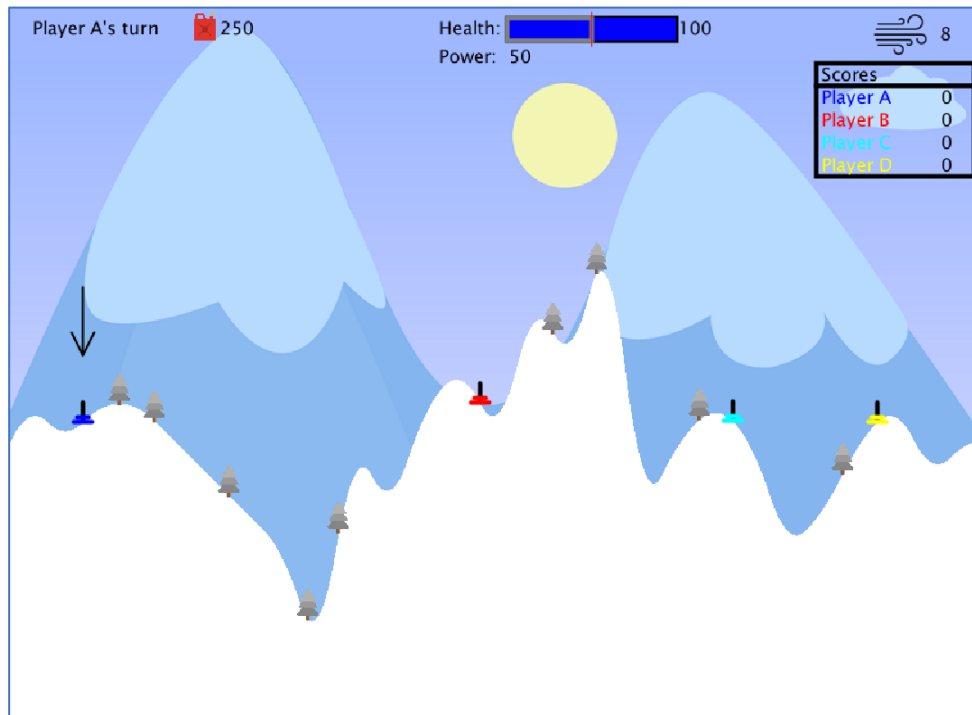
*Figure 1.
Player A (blue) has their turn first. The arrow pointing down to their tank exists for 2 seconds to remind them where they are – this appears at the beginning of each player's turn.*

There are 5 main types of characters that could be present in the file:

- X – denotes the terrain height. This can change during gameplay when hit by projectiles. To smooth out the terrain, see the section on page 3.
- Letters (A,B,C,D,E, etc.) – starting position of human players. The order of player turns, and the order in the scoreboard, is alphabetical order.
- Numbers (0,1,2,3,4,5,6,7,8,9) – starting position of AI players (optional for extension, otherwise they can just be normal human players as well)
- T – location of trees. They are always present on top of the terrain – so if the terrain changes, so do any trees on top of it. The initial position of trees is randomised up to 30 pixels around its starting point.
- Spaces – empty space, just ignore it.

The level layouts are defined in files provided in the "layout" attribute of the JSON configuration file described below. Each level must have an associated layout file.

Note that the file does not need to contain exactly 28x20=560 characters. It may have less than this if they are not necessary (such as spaces at the end of a line, or missing lines at the bottom). In such situations, your program should still work, and just assume those are empty spaces.

```
level1.txt ✖
 1
 2
 3
 4
 5
 6
 7                         X
 8                     T
 9                     X
10                   B   T
11       TT         X   X     C
12     XAXX         X         TX     DX
13      X   XT    X     X       X X   X   X
14   X       X                       T   X
15         XT   X           X       X
16         X                     X
17
18             T
19             X
```
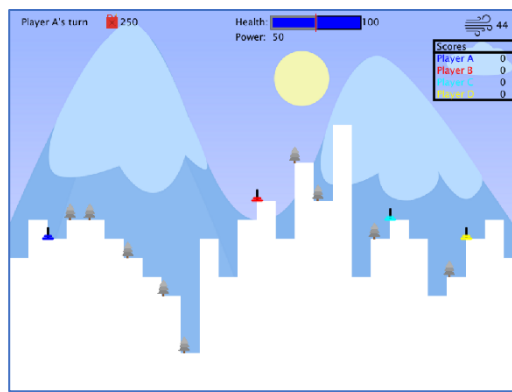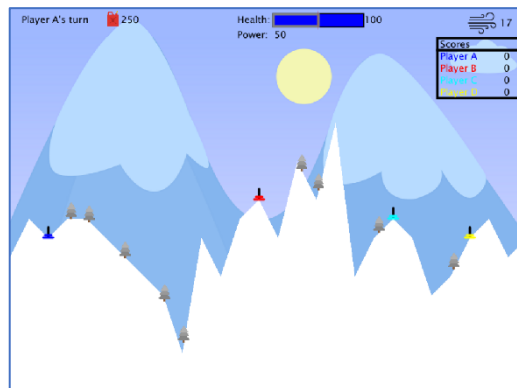
How to smooth the level terrain:

The terrain should comprise of a smooth curve that is formed from computing the moving average of 32 values twice. See below:
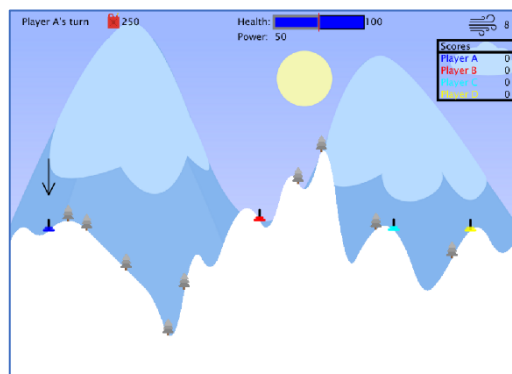
Step 1.
Terrain directly from file without any smoothing (each character is 32 pixels wide)

Step 2.
Moving average of 32 values, once.

Step 3.
Moving average of 32 values, again.

```
1   {
2       "levels": [
3           {
4               "layout": "level1.txt",
5               "background": "snow.png",
6               "foreground-colour": "255,255,255",
7               "trees": "tree2.png"
8           },
9           {
10              "layout": "level2.txt",
11              "background": "desert.png",
12              "foreground-colour": "234,221,181",
13          },
14          {
15              "layout": "level3.txt",
16              "background": "basic.png",
17              "foreground-colour": "120,171,0",
18              "trees": "tree1.png"
19          }
20      ],
21      "player_colours": {
22          "A": "0,0,255",
23          "B": "255,0,0",
24          "C": "0,255,255",
25          "D": "255,255,0",
26          "E": "0,255,0",
27          "F": "random",
28          "G": "random",
29          "H": "random",
30          "I": "random",
31          "0": "0,0,0",
32          "1": "0,0,0",
33          "2": "0,0,0",
34          "3": "0,0,0",
```

Config

The config file is in located in config.json in the root directory of the project (the same directory as build.gradle and the src folder). Use the simple json library to read it. Sample config and level files are provided in the scaffold.

The map layout files will also be located in the root directory of the project. However, sprites or images such as the background, and trees will be located in the resources folder (src/main/resources/Tanks/) or (build/resources/main/Tanks/).

In addition to providing the filenames of these files, the configuration file also provides the colours that should be used for the foreground of each level, and players. The format is "R,G,B" containing the red, green and blue components of the colour (0-255) respectively. If it is "random", then choose the colour randomly.

For each level, the following properties are provided in the config:

- layout: the level file containing the characters determining the initial terrain, player positions, and trees (see pages 2 and 3 above).
- background: the image which should be displayed in the background, behind the terrain and everything else.
- foreground-colour: the colour to render the terrain in for this level.
- trees (optional): The sprite image to use for trees in this level. If not specified, your program should not crash (either assume the level contains no trees so don't render any, or use a default sprite such as tree1.png).

Tanks

Initial positions of tanks are provided in each level's layout file. Each player can control the tank's turret movement, move it across the terrain, or increase and decrease the power level (the key must be held for these actions). Once they fire a shot (pressing the spacebar), their turn ends. A summary of the player input actions are in the table below:

Keyboard input	Action	Rate of change
UP arrow	Tank turret moves left	+3 radians per second
DOWN arrow	Tank turret moves right	-3 radians per second
LEFT arrow	Tank moves left across terrain	-60 pixels per second
RIGHT arrow	Tank moves right across terrain	+60 pixels per second
W	Turret power increases	+36 units per second
S	Turret power decreases	-36 units per second
SPACEBAR	Fire a projectile (ends turn)	

Note: the rate of change should be continuous (ie. per frame) and does not have to be exact, but is an indication to guide you in optimising user experience.

Tanks can only move across the terrain as long as they have fuel (denoted in the top right corner). Movement consumes 1 unit of fuel per horizontal pixel moved. Tanks start with 250 initial fuel in each level.

Tanks have initial health of 100, and power of 50, set at the beginning of each level. This is displayed in the HUD bar at the top. Power determines the speed (distance trajectory) of projectiles, and can never exceed the tank's remaining health. Once the tank's health reduces to 0, it explodes with an explosion radius of 15. If a tank goes below the bottom of the map, it explodes with a radius of 30.

How to change the turret's position using the angle from the vertical:
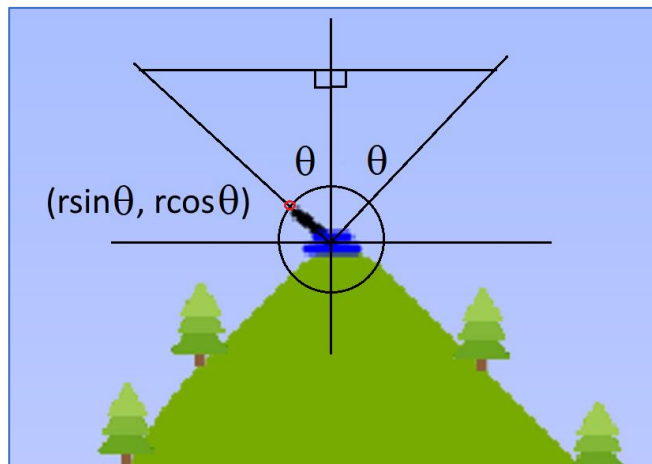

Figure 7. (left)
Recall the parametric form of a circle $(r\sin\theta, r\cos\theta)$. In this case, the turret length should be 15 pixels (r=15).

Note that the coordinate system in computer graphics has +x values going from left to right, and +y values going top-down, which is different from what you are familiar with in maths (x axis is the same, but in maths the y axis +ve values go from bottom to top).

Parachutes

A parachute is deployed for a tank when it is hovering in midair, due to the terrain below it having been destroyed by a projectile. Each player has 3 parachutes per game. If using a parachute, the tank descends at a rate of 60 pixels per second and sustains no damage. If no parachutes are available, it descends at a rate of 120 pixels per second and sustains damage of 1hp for each pixel of height (this is added as score to the player who fired the projectile that caused the terrain to be destroyed).

Projectiles

Projectiles spawn from the tank's location and follow a trajectory that is determined by the tank turret's vector. The magnitude of this vector is determined by the tank's power level at the time the missile is fired. The power level cannot exceed the tank's health. The minimum magnitude of the projectile's initial velocity is 1 pixel (power level 0), and the maximum is 9 pixels (power level 100). You can assume acceleration due to gravity is a constant rate of 3.6 pixels per second (applied per frame).

The player's turn ends after they fire a projectile (press spacebar).

Explosions

When a projectile makes contact with the terrain, it causes an explosion. The default explosion radius is 30 pixels. Any tanks within the radius will sustain up to 60 hp lost depending on how close they are to the impact site, reducing linearly the further the distance away. For example, with radius=30, a tank which is distance 15 pixels away (half the radius) from the impact will sustain only 30hp damage. A tank which is 10 pixels away will sustain 40hp damage, and a tank which is 20 pixels away will sustain 20hp damage. A tank which is 31 pixels away will sustain no damage since it's outside the radius.
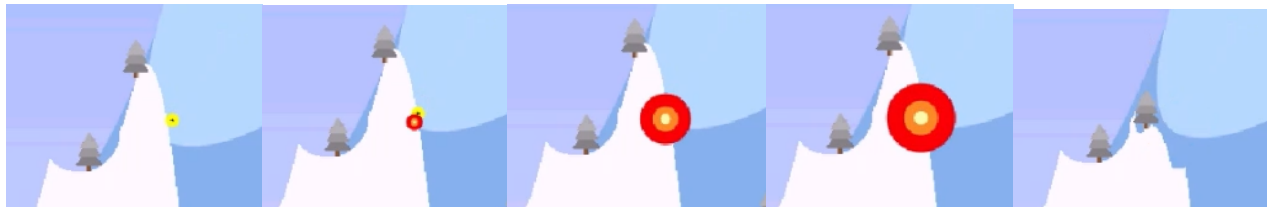
The animation (shown above) comprises of 3 concentric circles that expand continuously over a period of 0.2 seconds. During this time:

- Red circle expands from radius 0 to explosion radius (in this case 30px)
- Orange circle expands from 0 to 50% of the explosion radius (in this case 15px)
- Yellow circle expands from 0 to 20% of the explosion radius (in this case 6px)

The terrain will also be destroyed in a radius of 30 pixels around the impact site. Note that terrain cannot be floating, so if the impact is on the side of a hill, terrain above falls to fill the crater. Any trees on the terrain also fall to remain on top of the terrain. Trees are not destroyed by impacts.

Wind

Wind is a force that acts on projectiles in the air, in the horizontal (x) direction. It can blow either left or right, denoted by the different icons (shown above) and has a magnitude. The force imparted by wind is an acceleration of approximately $w*0.03$ pixels per second.

Wind is initially a random value between -35 and 35 (where negative values represent wind blowing to the left, and positives values are blowing to the right). After each turn, it changes by a random value from between -5 to 5 (inclusive of both).

Wind is displayed in the top right corner of the screen.

Powerups

During their turn, players can purchase powerups by using their score as currency. Powerups can only be purchased when the player can afford it, otherwise the action does nothing.

INFO1113 students must implement the following powerups:

- Repair kit (key: r, cost: 20) – repairs the player's tank by increasing health by 20 (maximum health is 100).
- Additional fuel (key: f, cost: 10) – increase the player's remaining fuel by 200.

COMP9003 students must implement the following powerups:

- Additional parachute (key: p, cost: 15) – increase the player's remaining parachutes by 1.
- Larger projectile (key: x, cost: 20) – the next shot fired by this player will have double the radius (60 instead of 30). Projectile damage is not increased, but will affect tanks within the radius according to the previous rules (proximity distance). A visual indication needs to be shown in the GUI / HUD that the next projectile to be fired will be a larger one.

Scoreboard

The scoreboard is persistent across levels. When a projectile fired by a player damages another player's tank, the player who fired that projectile gains score equal to the hp damage caused. A player does not gain score when a projectile they fire damages their own tank.

The scoreboard is ordered by player character (alphabetical order) and displayed in the top right corner. The text colour for each player is the same as their tank colour.

Level End and Game End

A level ends when there is only one tank remaining in play. The next level is loaded after 1 second, or whenever this tank fires a shot. The sequence of levels is the order they are provided in the configuration file.

When the last level ends, the game ends – display the player with the highest score as the winner (eg. "Player A wins!") and then in the centre of the screen, display the final scores in a box filled with a colour corresponding to the player's colour, and slightly lighter and transparent. The scores (and the corresponding players who achieved them) should be displayed in descending order (highest to lowest score), with a delay of 0.7s between when each one appears.

The user may press 'r' to restart the game.

Application

Your application will need to adhere to the following specifications:

- The window must have dimensions 864x640
- The game must maintain a frame rate of 30 frames per second.

- Your application must be able to compile and run using Java 8 and gradle run. Failure to do so, will result in 0% for Final Code Submission. Later versions of Java may work, but you should not use any newer Java language features.
- Your program must not exhibit any memory leaks.
- You must use the processing library (specifically processing.core and processing.data), you cannot use any other framework for graphics such as javafx, awt or jogl

You have been provided a /resources folder which your code can access directly (please use a relative path). These assets are loadable using the *loadImage* method attached to the PApplet type. Please refer to the processing documentation when loading and drawing an image. You may decide to modify these images if you wish to customise your game. You will be required to create your own sprites for any extensions you want to implement.

Extension

The extension is worth 2 marks maximum. For an extension, you can choose to implement one of the following:

- Additional powerups (eg, if you are doing INFO1113, you can implement the COMP9003 powerups as an extension, and vice versa). Or powerups of your own choosing, such as:
  - Shield (key: h, cost: 20) – protect your tank from taking damage the next time it's hit. After it is gets hit, the shield is destroyed (absorbed to sustain the blast). This must be indicated in the GUI (like with a blue transparent bubble around the tank).
  - Teleport (key: t, cost: 15) – select a location on the map to immediately move your tank there.
- Computer-controlled players. A targeting AI which can compute the trajectory of a projectile and adjust its aim to focus on hitting a player.
- Unlimited levels with randomly generated level terrain and player positions / trees, and randomised background
- Sound effects

OR, a feature you come up with which is of a similar or higher level of complexity (ask your tutor)

Please ensure you submit a config and level layout file with the features of your extension (if the extension required changes to the level or config files), and ensure the extension doesn't break any of the default behaviour. Also, describe your extension functionality in the report.

Marking Criteria (20%)

Your final submission is due on Sunday 12 May 2024 at 11:59PM. To submit, you must upload your build.gradle file and src folder to Ed. Please also include sample config and layout files that you have tested with your program to ensure it works. Do NOT submit the build folder (unless you only include the build/reports/ folder which contains the results of your testing and code coverage). Ensure src is in the root directory with the other files, and not part of a zip, then press MARK. Submit your report and UML to canvas.

A demo of your assignment will be conducted during labs in week 12 where you will demonstrate the features you have created to your tutor.

Final Code Submission and Demo (12%)

You will need to have implemented and satisfied requirements listed in this assignment. Make sure you have addressed the following and any other requirements outlined previously.

- Window launches and shows map layout correctly.
- Terrain display is correct – smooth curve, and matches level layout file
- Player starting positions and tree positions are correct, and correct tree type is rendered
- Configuration file is correctly read in – initial values for game state are used from config (player colours, level backgrounds and foregrounds)
- Multiple players take turns at controlling their own tank – turns are processed in alphabetical order according to config file
- Tank turret can be controlled with up/down arrow keys and angle changes (3 rad/s)
- Tank can move across terrain with left/right arrow keys (speed is 60px/s)
- Tank turret power can increase/decrease with 'w' and 's' keys
- Tank fires a projectile according to the turret's trajectory when the user pressed spacebar
- Projectile moves through the air and is affected by gravity and wind
- When a projectile collides with terrain, it explodes with a radius of 30 pixels
- Explosion is animated correctly
- Explosion causes damage to nearby tanks in a linearly decreasing proportion based on distance
- Tanks deploy parachute if available to fall slowly if the terrain underneath is destroyed, otherwise they fall faster (2x speed) and incur damage.
- Can use powerups
    - For INFO1113: Repair kit, and extra fuel
    - For COMP9003: Additional parachute, and larger projectile
- When tank reaches 0 health it is destroyed with an explosion, and that player's turn is skipped for the current level.
- GUI elements:
    - Health bar which displays changes to tank's health
    - Fuel indicator shows how much fuel is remaining
    - Parachutes indicator shows parachutes remaining
    - Power as a proportion of remaining health bar
    - Wind indicator that changes icon depending on direction of wind
    - Text for current player's turn, and arrow that appears at the beginning of the player's turn and lasts for 2 seconds, pointing to their tank
- Scoreboard tracks scores correctly
- Scoreboard is rendered with correct player colours
- The current level ends, and the next level is loaded when only 1 tank remains in play (after 1 second, or as soon as the player fires a shot).
- When all levels are complete, the player with the highest score is declared to be the winner and the final scores are displayed larger in the centre of the screen in a box whose background colour matches that of the winning player and is slightly transparent
- Final scores are displayed in descending order with a delay of 0.7s between each subsequent player
- Once the game has ended, a player can restart the game by pressing 'r'
- Ensure that your application does not repeat large sections of logic
- Ensure that your application is bug-free

Testcases (3%)

During development of your code, add testcases to your project and test as much functionality as possible. You will need to construct unit test cases within the src/test folder using JUnit. To test the state of your entities without drawing, implement a simple loop that will update the state of each object but not draw the entity.

Ensure your test cases cover over 90% of execution paths (Use jacoco in your gradle build) – average of branches and instructions. Ensure your test cases cover common cases. Ensure your test cases cover edge cases. Each test case must contain a brief comment explaining what it is testing. To generate the testing code coverage report with gradle using jacoco, run "gradle test jacocoTestReport".

Design, Report, UML and Javadoc (3%)

You will need to submit a report that elaborates on your design. This will include an explanation of any object-oriented design decisions made (such as reasons for interfaces, class hierarchy, etc) and an explanation of how the extension has been implemented. This should be no longer than 500 words. This report will be submitted through Canvas.

You will need to submit a UML diagram in PDF form to Canvas to provide a brief graphical overview of your code design and use of Object Oriented Principles such as inheritance and interfaces. Markers will use this to determine whether you have appropriately used those principles to aid you in your design, as well as figure out whether more should have been done. A general guideline is that markers will be looking for some use of inheritance or interfaces, how extensible the code is, and penalising repeated code. Note that you should not simply use a UML generator from an IDE such as Eclipse, as they typically do not produce diagrams that conform to the format required. We suggest using software such as LucidChart or draw.io for making your diagrams.

Your code should be clear, well commented and concise. Try to utilise OOP constructs within your application and limit repetitive code. The code should follow the conventions set out by the Google Java Style Guide. As part of your comments, you will need to create a Javadoc for your program. This will be properly covered in week 11 but the relevant Oracle documentation can be found here.

Report, UML and OO design: 2%
Javadoc, comments, style and readability: 1%

Extension (2%)

Implement an extension as described above. Partial marks may be awarded if you choose a more limited extension or it is partially completed. Please specify what extension you decided to implement within your report, and show it during your demo in week 12.

Suggested Timeline

Here is a suggested timeline for developing the project. Note that it is released on April 8 (start of week 7) and due May 12 (end of week 11).

Week 7: Familiarise yourself with gradle and processing, utilising the processing Javadoc and week 8 supplementary lecture. Identify opportunities to utilise Object Oriented Design principles such as inheritance and interfaces and begin to plan a design for the codebase with regards to the classes that you will need to make. Make a rough UML diagram for your design that you can base your codebase from.

Week 8: Begin writing the actual code for the program. Start small, for example by initially creating the map layouts and terrain, then gradually add more like player tanks. At the end of the week, you should have loading in the map and tank movement finished, as well as some sprite management. If confident, use Test Driven Development (writing test cases at same time as writing the code). Conduct a large amount of user testing to ensure the initial mechanics work as expected.

Weeks 9-10: Develop more gameplay features, such as the projectile movement, wind, collisions with terrain and tanks, powerups and scoreboard. Sprite management should be streamlined at this point. You should have a fairly high code coverage for your test cases at this stage. If you are noticing any questionable design decisions, such as God classes or classes that are doing things they logically should not be doing, this is the time to refactor your code. Think about what extension you want to make and start to implement it.

Week 11: Finish developing the remaining features for your program, notably the configuration file, GUI enhancements, timers and level progression. Additionally, finish writing your testing suite. Create the UML and Javadoc for the program. Fix any remaining bugs that your code exhibits. Submit your code to Ed (by uploading the entire project and pressing MARK) and submit your UML to Canvas in PDF form.

Week 12: Demonstrate the completed program to your tutor during the week 12 lab. They will check each criteria item has successfully been completed, and may ask you questions about how you implemented it to test your understanding.

Academic Declaration

By submitting this assignment you declare the following:

I declare that I have read and understood the University of Sydney Student Plagiarism: Coursework Policy and Procedure, and except where specifically acknowledged, the work contained in this assignment/project is my own work, and has not been copied from other sources or been previously submitted for award or assessment.

I understand that failure to comply with the Student Plagiarism: Coursework Policy and Procedure can lead to severe penalties as outlined under Chapter 8 of the University of Sydney By-Law 1999 (as amended). These penalties may be imposed in cases where any significant portion of my submitted work has been copied without proper acknowledgment from other sources, including published works, the Internet, existing programs, the work of other students, or work previously submitted for other awards or assessments.

I realise that I may be asked to identify those portions of the work contributed by me and required to demonstrate my knowledge of the relevant material by answering oral questions or by undertaking supplementary work, either written or in the laboratory, in order to arrive at the final assessment mark.

I acknowledge that the School of Computer Science, in assessing this assignment, may reproduce it entirely, may provide a copy to another member of faculty, and/or communicate a copy of this assignment to a plagiarism checking service or in-house computer program, and that a copy of the assignment may be maintained by the service or the School of Computer Science for the purpose of future plagiarism checking.