

INFO1113 / COMP9003

Assignment

Due: 22 October 2023, 11:59PM AEST

This assignment is worth 20% of your final grade.

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, the player must be able to place and upgrade towers strategically on a map to prevent enemies from reaching the wizard's house. Mana is the currency of the game used to buy and upgrade towers. Each time an enemy reaches the wizard's house, mana must be expended to banish it. The goal is to survive all waves of enemies without letting the wizard's mana run down to 0.

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.

Board

The board consists of a grid of tiles 20x20. Each tile is 32x32 pixels, so the total is 640x640 pixels. However, there are 40 pixels at the top reserved for information such as the timer indicator for when the current wave will end or the next one will start, and the wizard's current mana. There is also 120 pixels on the right sidebar for the gameplay action buttons and information about upgrade cost. The window size is therefore 760x680.

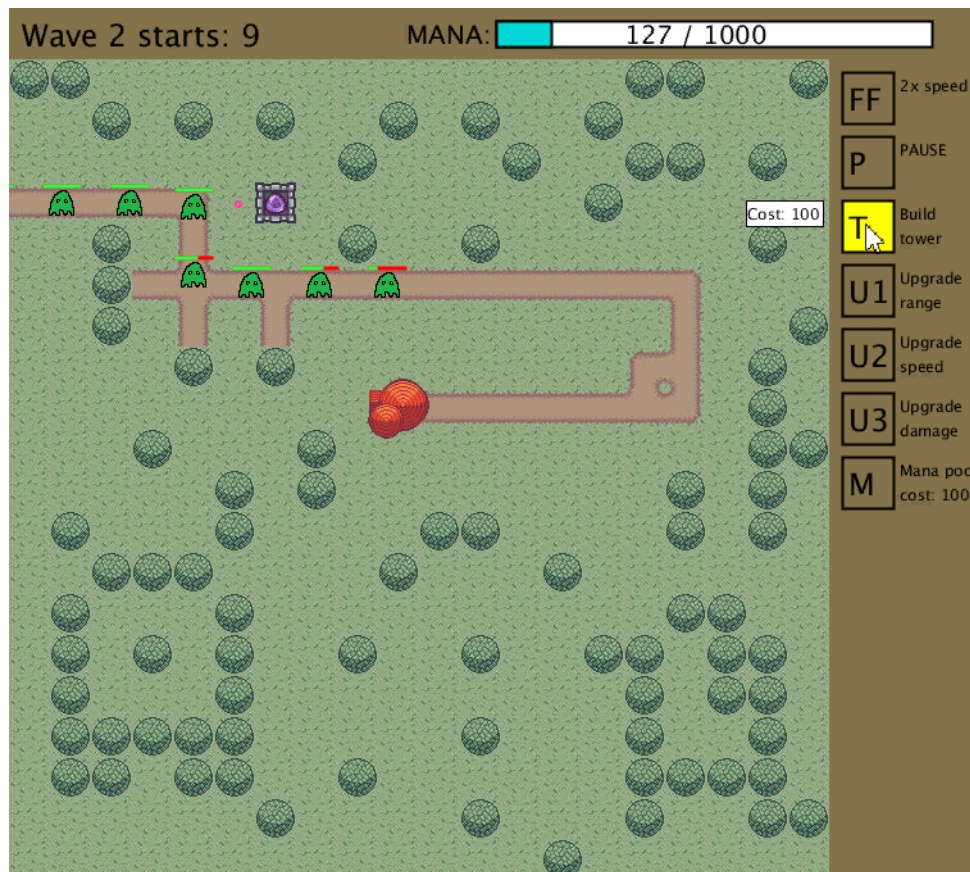


Figure 1.
The cost tooltip is shown when the user hovers over the “Build tower” or “Mana pool” action buttons with their mouse.

There are 4 main types of tiles:

- Grass – towers can be placed here
- Shrub – towers cannot be placed here
- Path – enemies traverse the path to try and reach the Wizard's house
- Tower – it shoots fireballs at enemies within its range
- Wizard's house – there can only be one on each map. If enemies reach here, they are banished by the wizard, causing loss of mana equivalent to the monster's remaining HP and then the monster is respawned on the map.

The initial map layout is defined in a file named in the “layout” attribute of the JSON configuration file described below.

Config


```


1  {
2    "layout": "level2.txt",
3    "waves": [
4      {
5        "duration": 8,
6        "pre_wave_pause": 0.5,
7        "monsters": [
8          {
9            "type": "gremlin",
10           "hp": 100,
11           "speed": 1,
12           "armour": 0.5,
13           "mana_gained_on_kill": 10,
14           "quantity": 10
15         }
16       ]
17     },
18     {
19       "duration": 5,
20       "pre_wave_pause": 10,
21       "monsters": [
22         {
23           "type": "gremlin",
24           "hp": 150,
25           "speed": 1.0,
26           "armour": 0.7,
27           "mana_gained_on_kill": 70,
28           "quantity": 25
29         }
30       ]
31     }
32   ],
33   "initial_tower_range": 96,
34   "initial_tower_firing_speed": 1.5,
35   "initial_tower_damage": 40,
36   "initial_mana": 200,
37   "initial_mana_cap": 1000,
38   "initial_mana_gained_per_second": 2,
39   "tower_cost": 100,
40   "mana_pool_spell_initial_cost": 100,
41   "mana_pool_spell_cost_increase_per_use": 150,
42   "mana_pool_spell_cap_multiplier": 1.5,
43   "mana_pool_spell_mana_gained_multiplier": 1.1
44 }

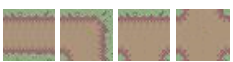
```

The config file is 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 file is also located in the root directory of the project. The layout file will contain a grid of text characters, where each character represents the tile that should initially be in that position on the map.

- Spaces are Grass 

- S = Shrub 

- X = Paths 

- W = Wizard's house 

Note that the wizard's house is 48x48 pixels and should be centred within its tile. It is rendered on top of nearby tiles and any enemies.

A function has been provided to you in the scaffold code to rotate the path tiles in order to create the full set of possible path tiles that may be needed.

Waves

The configuration file contains a list of waves in the "waves" attribute. Each wave occurs in the sequence it is provided there. During a wave, monsters spawn randomly on paths leading outside the map. A wave is defined in the config file with the following properties:

- **pre_wave_pause**: Number of seconds to wait before the wave begins. This begins counting after the duration of the previous wave has ended (or in the case of the first wave, immediately when the game starts).
- **duration**: Seconds during which this wave spawns monsters. The monsters will spawn randomly in equal intervals. For example, if the duration is 5 seconds, and there are a total of 25 monsters in this wave, there should be $5/25 = 0.2$ seconds between each monster spawn, or at 60 frames per second (FPS), 12 frames in between monster spawns.

- **monsters:** A list of monster types to spawn during this wave. The wave may contain multiple different types of monsters, and will choose a type at random when spawning each one (as long as there is quantity of that type remaining for this wave). The total number of monsters in the wave is the sum of quantities for all types provided in this list.

The top left corner of the GUI should indicate the time in seconds until the next wave begins with “Wave <x> starts: <s>”. If the last wave has begun, display nothing.


Once the last wave has been completed and all monsters are dead, the player wins the game.

Monsters

Monsters initially spawn outside the map on paths that join with the edge of the map. The spawn point is determined randomly out of all available locations. They travel on paths towards the Wizard’s house. The below configuration options define a monster type (as part of a wave):

- **type:** the sprite image to use for the monster (png file extension)
- **hp:** amount of max hit points (initial health) the monster has
- **speed:** how fast the monster’s movement is in pixels per frame
- **armour:** percentage multiplier to damage received by this monster
- **mana_gained_on_kill:** How much mana the wizard gains when this monster is killed

A monster’s current hp should be shown as a proportion of their original hp above their sprite, with a green and red bar (see image on page 2).

When a monster dies, there is a death animation with each image lasting 4 frames: 

Towers

Towers can be placed by the player in empty grass tiles, by first activating the tower action (either clicking on the T button in the right sidebar, or pressing the key ‘t’), then clicking on the space they want the tower to be placed. The following configuration options are relevant for towers:

- **tower_cost:** The cost in mana to place a tower. Depending on if upgrades are selected, the initial tower cost may have +20 (one upgrade), +40 (two upgrades) or +60 (three upgrades).
- **initial_tower_range:** The radius in pixels from the center of this tower within which it is able to target and shoot at monsters.
- **initial_tower_firing_speed:** Fireballs per second rate that this tower shoots at.
- **initial_tower_damage:** Amount of hitpoints deducted from a monster when a fireball from this tower hits it. If the monster’s health reaches 0, then it dies.

Towers may be upgraded by the player by selecting the upgrade action for either range, speed or damage (either via clicking the option in the left sidebar, or pressing keys ‘1’, ‘2’ or ‘3’ respectively). Then, the player must click a tower to upgrade. Before clicking the tower, just by hovering over it, the cost of the upgrade will be shown in the bottom right corner of the screen, as below.

Multiple upgrade options may be selected at the same time. If the wizard doesn’t have enough mana for all of them, then only the ones they can afford will be purchased (in order as they appear in the actions column: range, speed, then damage). The player cannot cause themselves to lose on purpose by spending more mana than they have.

The player may also build towers with initial upgrades of level 1 by having the upgrades action selected together with the “build tower” action when placing the tower.

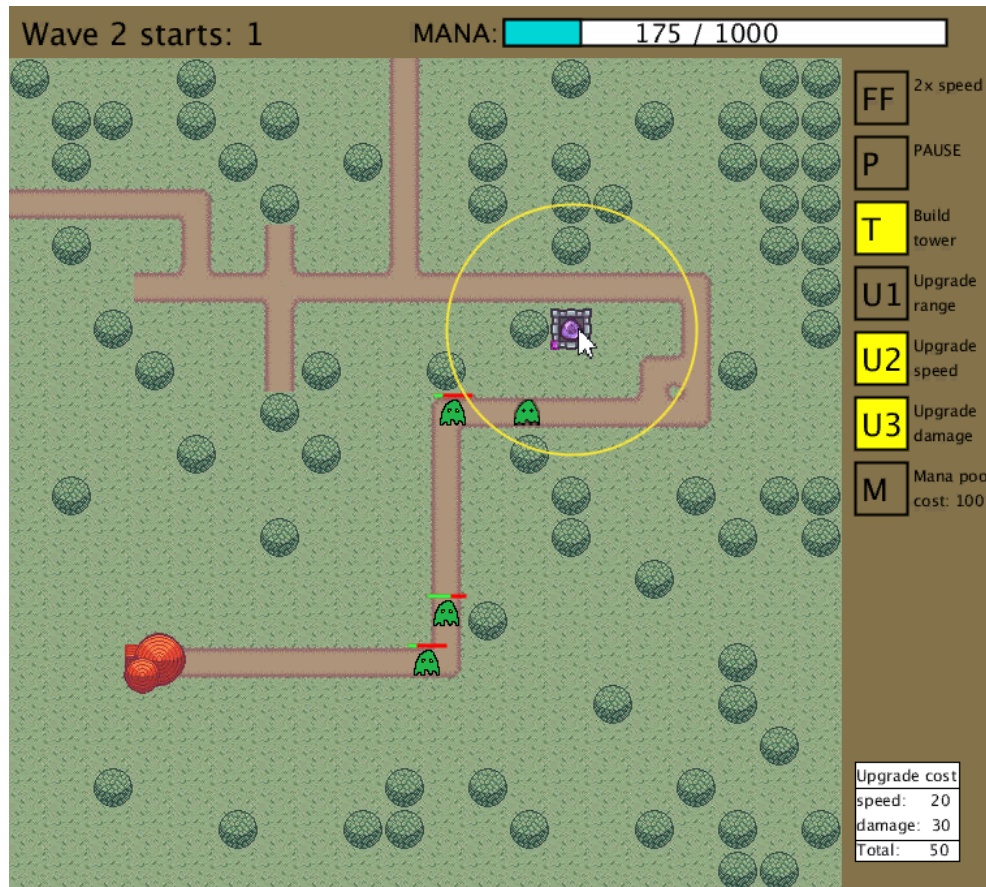


Figure 2.

The upgrade cost is shown only for upgrade options that have been selected. In this case, the tower was already upgraded to damage lvl 1, so the upgrade cost is 20 for speed lvl1 and 30 for damage lvl2.

Note that the tower's range radius is also highlighted in yellow when the player's mouse hovers over it.

Examples of tower upgrades and how they appear on the tower image:

| | |
|--|--|
| | Range level 1 |
| | Damage level 1 |
| | Speed level 1 |
| | Speed level 3 |
| | Range level 3 |
| | Range level 2 and damage level 3 |
| | Range, speed and damage all lvl 1 |
| | Level 1 in range and damage, level 2 in speed. |
| | Range level 3, speed level 2 |
| | Range, speed and damage all lvl 2 |

Tower upgrade cost:

The upgrade cost depends on the level. It is 20 mana for level 1, then increases by 10 for each subsequent level. This means level 2 costs 30, level 3 costs 40, etc. This cost is for each upgrade, so to upgrade to level

Upgrade symbols:

The range upgrade level is denoted by magenta “O”s at the top of the image.

The speed upgrade level is denoted by the thickness of a blue border in the centre of the image.

The damage upgrade level is denoted by magenta “X”s at the bottom of the image.

Once level 1 of all upgrades is reached, the tower becomes orange (and no longer displays the indicators for level 1, only levels 2+). Similarly, if level 2 is reached in all upgrades, then it becomes red (and will only use the individual symbols X, O and blue border for levels 3+).

1 fully in all of range, speed and damage, would cost 60 mana. Then to upgrade to level 2 fully would cost another 90 mana.

Tower upgrade effects for each upgrade level:

- Range upgrade: tower range radius increases by 1 tile (32 pixels)
- Speed upgrade: Firing speed increases by 0.5 fireballs per second
- Damage upgrade: Damage dealt increases by half of `initial_tower_damage`

Fireballs •

Fireballs are created by towers to shoot an enemy currently within their range. The choice of which enemy to shoot is arbitrary. Fireballs move at a speed of 5 pixels per frame. A fireball originates at the centre of the tower and targets the centre of the enemy it was fired at. When it reaches the centre of that enemy, it damages the enemy.

Wizard

The Wizard's house is the destination for monsters. When a monster reaches the wizard's house, the monster is 'banished' – mana equal to the monster's remaining HP is expended in order to make the monster respawn and navigate the path again.

The Wizard's house should be rendered on top of other tiles and any monsters.

Figure 3.

Note that the yellow tower range radius indicator (when the mouse hovers over a tower) should appear behind the top bar of the GUI that contains the mana indicator and wave timer, as well as behind the right bar containing the action buttons.



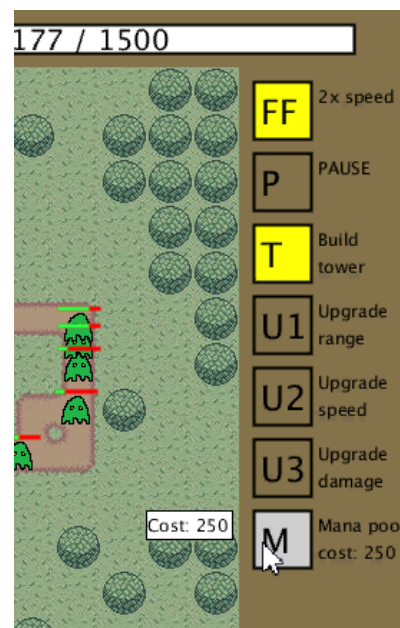
Mana

The wizard's mana is visible in an aqua bar at the top of the screen, as shown in images previously. There is a mana cap – additional mana is not gained if the cap is reached. The initial values of these are set by the configuration options `initial_mana` and `initial_mana_cap`. A small trickle of mana is also gained as time passes, specified in the configuration attribute `initial_mana_gained_per_second`.

Mana Pool Spell

The mana pool spell is a special action that can be done at any time to increase the wizard's mana cap and add a multiplier to mana gained from monster kills and the mana trickle. The following configuration options are relevant:

- `mana_pool_spell_initial_cost`: The initial cost of the spell, in mana.
- `mana_pool_spell_cost_increase_per_use`: How much the spell's cost should increase after each use.
- `mana_pool_spell_cap_multiplier`: The multiplier to use for increasing the mana cap when the spell is cast (its effect is multiplied when activated multiple times).
- `mana_pool_spell_mana_gained_multiplier`: The multiplier to use for increasing mana gained from killing monsters and the mana trickle when the spell is cast. Its effect is added if the spell is cast multiple times (eg. 1.1 is +10% bonus to mana gained, which would become +20% if the spell is cast twice, or +30% if cast 3 times).



Gameplay Actions

The gameplay actions available to the player are listed in the right sidebar. When the player hovers their mouse over one of these options (when unselected), it should turn grey. When selected, it should be highlighted yellow. If clicked while selected, it will be unselected. Multiple options may be selected at once. Each action has a corresponding key that also may be pressed to toggle the option. The actions are:

- Fast forward (FF) – key: f. The game will run at 2x speed – meaning anything that takes time is sped up, including monster movement, wave timers, tower firing speed, and mana increment. Unselecting this option will resume normal speed.
- Pause (P) – key: p. The game will be paused and in a frozen state. Unselecting this option resumes the game. While paused, the player is still able to build and upgrade towers or cast the mana pool spell.
- Build tower (T) – key: t. When selected, allows the player to click a grass tile on the map to place a tower there (provided they have enough mana). If they don't click a grass tile, or don't have enough mana, this selection has no effect. The cost should be displayed when the mouse hovers over this option. The cost is only deducted when the tower is placed though, not when this option is selected.
- Upgrades (U1, U2, U3) – keys: 1,2,3. Allows the player to upgrade towers. May also be combined with build tower.
- Mana pool spell (M) – key: m. When selected, will activate the Mana Pool spell (see above on page 6) and then immediately deactivate. Cost should be displayed when the mouse hovers over it, and in the text description.

Win and lose conditions

The game ends in a win for the player when all waves are over and all monsters have been defeated. Display "YOU WIN" on the screen.

The game ends in a loss when a monster reaches the wizard's house with a banishment cost (current HP) greater than the wizard's current mana balance. Display "YOU LOST" on the screen, along with an option to restart the game by pressing the 'r' key. Restarting should cause everything in the game to be reset to its original state.

Application

Your application will need to adhere to the following specifications:

- The window must have dimensions 760x680
- The game must maintain a frame rate of 60 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 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 sprites 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:

- New type of tower with special behaviour (eg. freeze enemies, poison, splash damage)
- New type of building construction (eg. walls or traps)
- New type of user gameplay action (eg. inventory containing items that can be purchased and dropped on monsters to cause damage to them)
- New game mode (specified in config or initial game start menu) with unlimited waves where monsters get progressively stronger
- Ability to save and load games to and from a save file
- Sound effects (partial marks only unless another extension is also attempted)
- Multiple levels specified in the config that are loaded after the current level is beaten

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, 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 22 October 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.
- Path display is correct
- Configuration file is correctly read in – initial values for game state are used from config
- Waves of monsters spawn according to the configuration
- Wave timing is correct
- Game action buttons respond to mouse hover and click. Key press also works as an alternative to toggle the selection.
- Can place a tower only in empty grass cells
- Mana cost is correctly deducted for purchases and insufficient mana disallows purchases
- Can upgrade towers
 - Indicator is shown correctly for different individual upgrades (speed, range, damage)
 - Tower image changes to orange and red when all of level 1 or level 2 upgrades are done
 - Upgrades cause noticeable changes in tower behaviour for range, speed and damage dealt
 - Towers can be built with some initial upgrades if the upgrade option is selected
- Enemies spawn correctly outside of the map and come from paths connected to the edge of the map (randomly chosen).
- Enemies move only on paths
- Enemy movement is smooth and correct speed
- Enemies accurately traverse junctions towards the Wizard's house
- Towers shoot fireballs at enemies within their range
- Fireballs are displayed correctly and move at the correct speed
- Fireballs cause loss of health to enemies, and when the monster's health reaches 0, it dies
- Death animation for monsters with each image lasting 4 frames
- If an enemy reaches the wizard's house, it is banished
- Banished monsters respawn correctly and cause a deduction of mana equal to their remaining hp
- Mana pool spell causes an increase to the mana cap and multiplies mana gained
- GUI elements:
 - Tooltip for upgrade cost
 - Tower range radius highlighted on hovering over it
 - Health bar displayed above enemies
 - Mana bar displayed correctly
- Fast forward doubles the game speed, and pause action freezes the game
- Game ends in a loss if wizard doesn't have enough mana to banish a monster
- Game ends in a win if all waves are complete and all enemies defeated
- 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) 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](https://google.github.io/styleguide/javaguide.html). 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](https://docs.oracle.com/javase/8/docs/technotes/guides/javadoc/).

| | |
|---|----|
| 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.

Suggested Timeline

Here is a suggested timeline for developing the project. Note that it is released on September 12 (start of week 7) and due October 22 (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 layout and tiles, then gradually add more elements. At the end of the week, you should have loading in the map and enemy 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 action buttons (adding and upgrading towers), shooting fireballs, mana display and the mana pool spell. 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, and timer for waves. 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.