



INFO1113

Assignment

Due: 23 October 2022, 11:59PM AEST

This assignment is worth 18% 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 avoid enemies that are moving around the map, and reach the exit to progress on to the next level.

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 (5d).

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

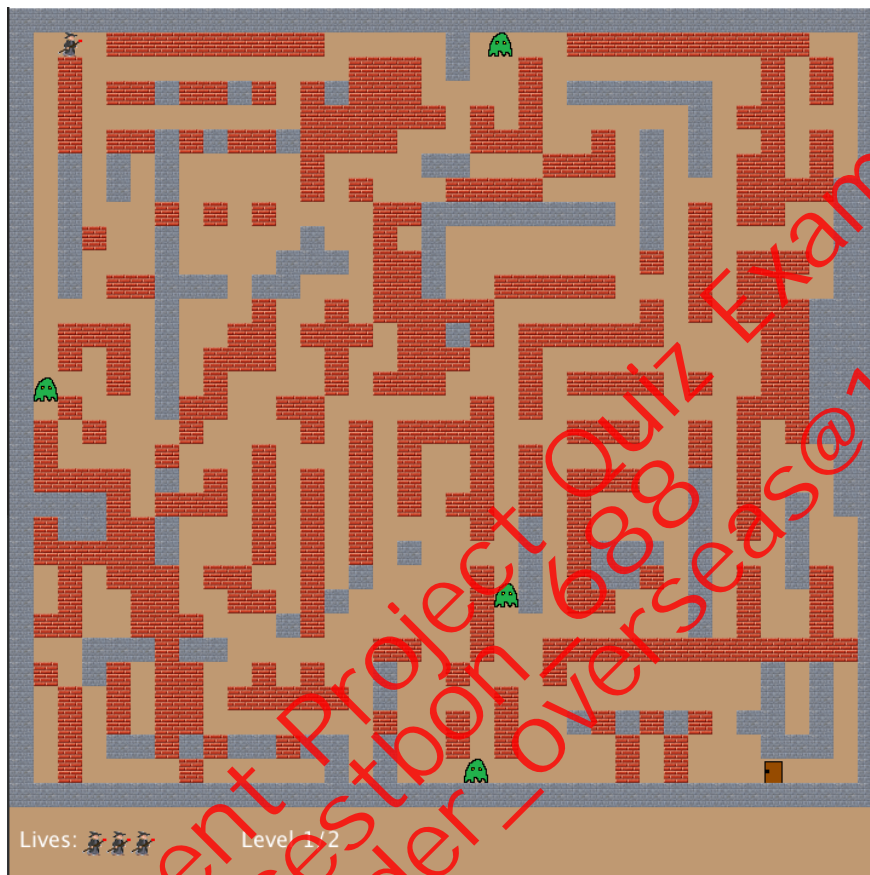
You have been given a scaffold which will help you get started with this assignment. You can download the scaffold onto your own computer and invoke gradle build to compile and resolve dependencies. You will be using the Processing library within your project to allow you to create a window and draw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

Map

The map consists of a grid of tiles 33x36. Each tile is 20x20 pixels, so the total is 720x660 pixels. The player always begins in the top-left corner of this grid. However, the bottom 60 pixels of the window are reserved for the information bar which contains text to display the current number of lives, spell cast cooldown bar, current level number, and timer remaining on the powerup's effect. The window size is therefore 720x720.



There are 3 main types of tiles:

- Stone wall
- Brick wall
- Exit

The map is always surrounded by a stone wall. The map layout is specified in a file named in the "layout" attribute of the level in the JSON configuration file described below.

Config



```

1 {
2   "levels": [
3     {
4       "layout": "level1.txt",
5       "wizard_cooldown": 0.3333,
6       "enemy_cooldown": 3.0
7     },
8     {
9       "layout": "level2.txt",
10      "wizard_cooldown": 2.0,
11      "enemy_cooldown": 1.0
12    }
13  ],
14  "lives": 3
15 }

```

The config file is located in config.json in the root directory of the project. Use the simple json library to read it. Sample config and level files are provided in the scaffold.

The config sample as shown to the left, contains the names of the level files. These are also located in the root directory of the project. The level files will contain a grid of text characters, where each character represents what should be in that tile cell.

- G is where gremlins should be placed 
- W is where the player starts from 
- E is the exit goal the player must reach to end the level and progress on to the next one.





Note that a map is valid if it has a bounding border of cement tiles, contains a starting point and exit, and is of the correct dimensions. (all maps used for marking will be valid, but you should write your own tests for invalid maps and handle them as you see fit).


The "wizard_cooldown" property of the level denotes in seconds the cooldown time between spell casts.

The "enemy_cooldown" property denotes in seconds the cooldown time between gremlins shooting slime.

Wizard

The player character is controlled using the arrow keys (up, down, left, right). Movement should be smoothly transitioning from one tile space to another. The player begins in the tile 'W' on the map layout. The user must be actively holding the movement key for movement to occur, otherwise movement stops (when it reaches the next whole tile). The wizard may only stop movement on a whole tile space, not part-way between tiles.

The wizard sprite should change depending on the direction they are facing:    

The wizard can shoot fireballs by pressing the space bar. Fireballs travel in the direction the wizard is currently facing, until they hit an object. Fireballs can destroy brick walls, triggering the following animated destruction sequence:  with each image lasting for 4 frames. This absorbs the fireball. After a fireball spell is cast, the wizard must wait for their mana to recharge as shown with a progress bar in the bottom right-hand corner of the screen. Different levels may make it more difficult or easier for the wizard to cast spells (cooldown is specified in the config per level).

The player's movement speed is 2 pixels per frame. Fireball speed is 4 pixels per frame.

Gremlins

Gremlin enemies are green mischievous figures. It has the character 'G' in the map layout. When hit by a wizard's fireball, it will disappear and respawn in another empty area of the map, at least 10 tiles radius away from the player (in the process, absorbing the fireball). Each gremlin throws slime projectiles in the direction of their current movement, with a frequency in seconds specified in the configuration JSON for that level. If the gremlin hits a wall with more than one possible new direction to go in, it will randomly choose a new direction but won't go back the way it just came.

If the wizard comes into contact with a gremlin or its slime, they lose a life and the level is reset to its original state. If the wizard's fireball hits a gremlin's slime, the slime absorbs the fireball, and in the process is itself vapourised.

Gremlin movement speed is 1 pixel per frame and slime projectile speed is 4 pixels per frame.

Powerups

A powerup is an item that can be collected by the player upon moving to that location on the board (player character collides with it directly). You should decide a new symbol to use in the map layout config to denote a powerup. Please be creative in designing the sprite for the powerup you decide to implement. The functionality could be any one of the following things:

- Make enemies slow down for a timed period
- Make the player speed up for a timed period
- Make the player invincible for a timed period. Show some visual indication on enemies or the player that this is the case – such as a border, or different colour.
- Make enemies freeze for a timed period
- Reduce spell cast (fireball) cooldown time for a timed period. Show some visual indication on the spell cast cooldown progress bar (eg. different colour).
- Increase fireball area of effect for a timed period

Please ensure that the duration of the timer remaining is made clear to the player. (maybe there is a counter in the top bar, with the name of the powerup's effect. Or, a progress bar like as in the spell cast cooldown). You should determine the time interval it lasts for (maybe around 10 seconds is reasonable). You may also choose to implement a sound effect when the powerup is collected, and choose to animate it.

The powerup should not spawn immediately when the level loads, but only after some delay interval (within 10 seconds). And when collected, it may respawn after another randomised delay interval, with this process continuing. You may choose to implement multiple powerups and choose which one to randomly spawn in. Note that the powerup system must be able to work with all map types.

Win and lose conditions

The current level is completed when the player reaches the exit. If there is another level, that level is then loaded with the player starting in the position defined in the map layout. The player retains the number of lives they had previously.

If there are no more levels and the player wins, display a screen saying "You win".

If the player loses all of their lives, display a screen saying "Game over".

Any key press from either the game over or win screens should restart the game.

Application

Your application will need to adhere to the following specifications:

- The window must have dimensions 720x720
- The game must maintain a frame rate of 60 frames per second.
- Your application must be able to compile and run on any the university lab machines (or Ubuntu VM) using gradle build & gradle run. Failure to do so, will result in 0% for Final Code Submission.
- Your program must not exhibit any memory leak.
- 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 the powerup and any extensions you want to implement.

Extension

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

- New enemy type with special behaviour
- New type of breakable tile that triggers some effect (enemies freeze, player becomes invincible, increased speed, reduces cooldown)
- New type of spell (projectile) to cast with a different key – please state which key somewhere in the GUI. It should have a different effect (eg. freeze enemies, larger area of effect but longer cooldown)
- Doors which teleport the player to another tile

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 present in the first level. Also, describe your extension functionality in the report.

Marking Criteria (18%)

Your final submission is due on Sunday 23 October at 11:59PM. To submit, you must upload your build.gradle file and src folder to Ed. Please also include a sample config and level files that you have tested with your program to ensure it works. Do NOT submit the build folder. Ensure src is in the root directory with the other files, and not part of a zip, then press MARK. Submit your report, UML and a video demo of your game which is no more than 40MB to Canvas (pdf and mp4 only).

Final Code Submission (10%)

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 beige background.
- Configuration file is correctly read in – number of lives, cooldown for levels
- Map loads and tiles are displayed in correct positions
- Player and enemies are correctly loaded in
- Player is controlled by input from arrow keys
 - Is there movement at all – in correct directions
 - Is the movement smoothly transitioning from one tile to the next at the correct speed
 - Movement requires key to be held
- Enemies move correctly – movement is smooth and correct speed
- Enemy or Player movement does not clip through brick or stone walls
- Enemies choose a random direction to proceed when hitting a wall, that was not the direction they came from (unless it's the only available direction)
- Enemies cause the player to lose a life upon collision
- Gremlins hit with fireballs respawn in an empty tile on the map at least 10 tiles away from the player
- Gremlins shoot slime projectiles every cooldown interval as specified in the config
- Slime projectiles cause the player to lose a life when hit
- Slime projectiles absorb fireballs when hit, and are vapourised
- Player shoots fireballs when spacebar is pressed
- Fireballs cause brick walls to be destroyed
- Brick wall destruction sequence is animated correctly with each image lasting 4 frames
- Fireball cooldown is shown as a progress bar in the bottom right corner of the screen
- Player respawns upon life lost and is stationary, level is reset to original state
- Powerups spawn in the defined location in level config, some time after the level loads (within 10s)
- Powerups respawn a while after collected (again, within 10 seconds)
- The powerup has one of the desired effects
- All useful info is displayed correctly in the topbar of GUI (lives remaining, current level number, timer remaining on powerup)
- The next level is loaded if the player reaches the exit
- The win screen is shown if Player reaches the goal in the final level
- The game over screen is shown if the player loses all lives
- 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 at least three levels in a class hierarchy (similar to how in Question 2 of the Week 5 tutorial, Teacher extends Employee which extends Person). 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, such as an additional powerup that results in an extra life.

Or another basic extension might be to add a sound effect to events such as collecting a power up, losing a life, etc. 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 13 (start of week 7) and due October 23 (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 and the player, and gradually add more elements. At the end of the week, you should have loading in the map and player 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 enemies, projectiles, collisions and powerups. 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, level system and lives management. 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.

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.