THE UNIVERSITY OF SYDNEY

INFO1113 Assignment 2

Due: Friday 20 November, 11:59PM AEST

This assignment is worth 15% of your final assessment

Task Description

You are working for a company called ArcadeRetro which recreates popular arcade games for modern audiences. The company is currently developing a game called **Waka Waka**. In the game, the player must guide Waka around the map, collecting all the fruit whilst avoiding enemy ghosts.

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. Additional requirements will be released after your milestone submission.

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.

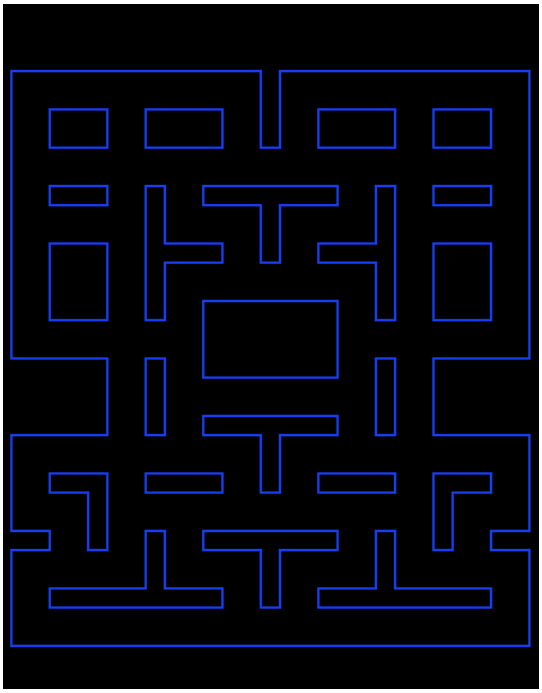
Working on your assignment

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 the following link. For the reading of configuration files, you will be using JSON.simple. You can access the documentation for the library here.

Gameplay

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

Map



The Map designates where Waka and ghosts are allowed to move. Laid out as a grid, entities can move along rows and columns where there is no wall (represented by a blue line). Maps are 36 rows tall, and 28 columns wide, however typically at least the top 3 and bottom 2 rows are left empty for additional UI elements. Each grid space is 16x16 pixels.

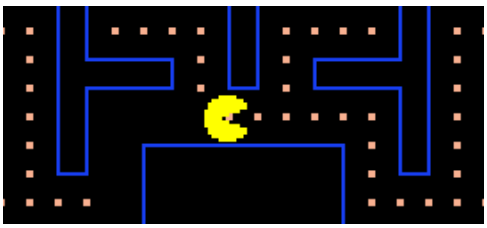
Map layouts are stored in text files, and the name of the map file can be found in config.json under the "map" attribute. Sample Map and config.json files can be found under resources on Ed. Map files store the maps as multidimensional character arrays, where each character represents what is in that cell. Note that a map is valid if it has a bounding border, a starting location for Waka, and at least one fruit. There is no minimum requirement for the number of Ghosts.

Character	Contents	Sprite
0	Empty cell	N/A
1	Horizontal wall	
2	Vertical wall	
3	Corner wall (up + left)	
4	Corner wall (up + right)	
5	Corner wall (down + left)	
6	Corner wall (down + right)	
7	Fruit	
p	Waka starting location	
g	Ghost starting location	

Waka

Waka is the player-controlled character in the game. Waka can move vertically and horizontally on the map and cannot pass through walls. Waka is controlled with the arrow keys (up, down, left and right).

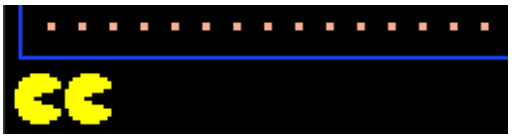
Unless colliding with a wall, Waka is always moving. The player can tell Waka what his next move should be, and Waka will turn accordingly when next possible. For example, if Waka were in the following position, moving right:



And the player was to press the down key, Waka would turn down when he reaches the end of the hall. If the player were to press the up key, Waka would make the next possible turn up. If the player were to press the left key, Waka would immediately turn around (Waka can always turn around). If the player were to press the right key, Waka would continue unchanged. Note that if the player presses multiple keys before Waka turns, the most recent key is used.

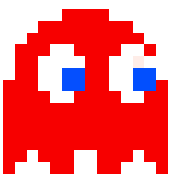
Waka moves at a constant speed, specified in the configuration file under the “speed” attribute. This speed will either be 1 or 2.

Also present in the configuration file is the number of lives Waka has. The field is stored under the “lives” attribute. The number of lives remaining is represented on the screen with sprites of Waka facing right at the bottom. When Waka runs out of lives, the game is over.



Waka has five sprites that can be rendered: one with a closed mouth and one open mouth sprite for each cardinal direction. Waka’s sprite should alternate between open-mouth and closed-mouth every 8 frames, and should also have the open-mouth sprite relate to the direction he is moving. For example, if Waka is moving upwards, Waka’s mouth should be pointing upwards.

Ghost



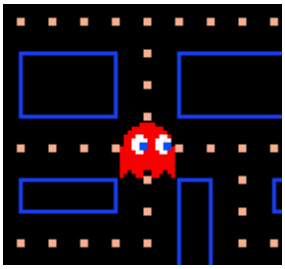
Ghosts are the antagonists of Waka Waka. Their goal is to prevent Waka from collecting all the fruit on the map. They do this by hunting him down and hitting him.

Like Waka, Ghosts can move horizontally and vertically on the map. They cannot pass through walls, however can pass through each other. Ghosts move at the same speed as Waka (specified in the game configuration file).

Ghosts have two modes they can be in: SCATTER and CHASE. These modes determine the behaviour of the Ghosts when they reach an intersection.

The durations of these modes are specified in the configuration file under the attribute “modeLengths”. The attribute is stored as an array of ints, representing the time (in seconds) each mode should be before alternating to the other, with the first mode being SCATTER. If the game has not ended by the time the array has been traversed through, the array restarts from the beginning in SCATTER mode.

When a Ghost reaches an intersection (a point at which it can make a turn), it determines its target location, and then moves in the direction of the adjacent grid space closest to the target, based on straight line distance. For example, if the target location for the Ghost in the image below was 3 rows up and 1 column left, the Ghost would move up. Conversely, if the target location were to be 1 row up and 3 columns left, the Ghost would turn left.



The mode the Ghost is in determines what the target location is:

- If the Ghost is in SCATTER mode, the target location is the closest corner of the map
- If the Ghost is in CHASE mode, the target location is the grid space occupied by Waka

Note that, unless trapped, a Ghost cannot turn around and move in the opposite direction to which it just moved. For example, if a Ghost has moved upwards to an intersection, it cannot then move down. Instead it must move in whichever of the other three cardinal directions is closest to the target location.

When a Ghost occupies the same grid space as Waka, all Ghosts and Waka return to their starting positions, and Waka loses a life. When Waka loses all of his lives, the game ends.

For testing purposes, ArcadeRetro has requested that you include a DEBUG mode for your game, which can be activated/deactivated by pressing the space key. While in DEBUG mode, a white line is drawn from each Ghost to their respective target location.

Fruit

Waka’s goal is to collect every fruit on the map. Waka does this by moving onto the grid space that the fruit occupies. When this occurs, the fruit is removed from the map. Fruit entities do not move, nor are they collected by ghosts. If Waka is hit by a ghost, fruits are not reset. Instead all fruit collected before the collision remain cleared from the map. Once Waka collects all fruit in the map, the game ends.

Application

Your application will need to adhere to the following specifications:

- The window must have dimensions 448x576
- 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, you cannot use any other framework such as javafx, awt or jogl.

Assets

Artists within the company have produced sprites for your game. You have been provided a /resources folder which your code access directly. These assets are loadable using the *loadImage* method attached the PApplet type. Please refer to the processing documentation when loading and drawing an image.

Marking Criteria (15%)

Your final submission is due on Friday 20 November at 11:59PM. To submit, you must submit your *build.gradle* file and *src* folder to Ed (by pressing MARK) and your report to Canvas.

Final Code Submission (5%)

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 black background
- Configuration file is correctly read in
- Map loads and walls are displayed
- Ghosts and Waka correctly loaded in
- Waka is controlled by input as described above
- Waka's sprite alternates between closed-mouth and open-mouth (with correct orientation)
- Ghosts clearly display different behaviours for SCATTER and CHASE
- Ghosts have the correct target locations and move towards them as specified above
- Pressing space activates/deactivates DEBUG mode
- Waka can collect fruit by passing through them
- Waka loses a life when he is hit by a Ghost
- Waka's lives are shown at the bottom of the screen
- Ensure that your application does not repeat large sections of logic
- Ensure that your application code exhibits good Object Oriented principles
- **Additional requirements will be announced after the milestone deadline which will need to be implemented**

Milestone (1%)

To ensure progress has been made within on your project, you will need to submit a working submission of your project by Monday 2 November by 11:59pm AEST. You should achieve the following as a minimum:

- Map and config file are read in
- Map is correctly rendered (including fruit)
- Waka is rendered and movable
- At least one Ghost is rendered

Note that marks for this component will be returned after the final due date. However, if you attend your scheduled lab in week 10, your tutor will be able to provide feedback on your current progress.

Test Cases (4%)

During development of your code, add test cases to your project and test as much functionality as possible. You will need to construct unit test cases within the src/test folder. 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.

Design, Report and Javadoc (4%)

You will need to submit a report that elaborates on your design (2%). 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.

As an appendix to your report, you need to include a compete UML diagram of your design (1%). Markers will refer to both your codebase and this diagram when judging your use of Object Oriented Design.

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 (1%). This will be properly covered in week 11 but the relevant Oracle documentation can be found here.

Extension (1%)

You have the choice of either adding one of the listed features to the program or creating your own feature (must first be approved by TA).

- Collectable soda-can that frightens ghosts and turns them invisible for a period of time
- 3D arcade machine, where the player can pause the game and walk around the arcade
- Be the ghost mode (requires AI for Waka)
- Online mode, where two players can play as Waka and Ms. Waka from different computers

Warning

Any attempts to deceive or disrupt the marking system will result in an immediate zero for the entire assignment. Negative marks can be assigned if you do not properly follow the assignment specification, or your code is unnecessarily or deliberately obfuscated.

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.