Assignment 1 - Pacman

Introduction

Welcome to the first of your assignments for COMP90041 - Programming and Software Development!

For this project, we will be implementing the game PACMAN on a console(terminal).

This program will be built in *two parts* (Assignment 1 and Assignment 2), with Assignment 1 building the groundwork for the final system in Assignment 2.

In *Assignment 1*, we will build an initial system implementation based on what we've learned about object-oriented programming so far. In *Assignment 2*, we will extend the system with additional features, and also refactor existing features with newly learned Object Oriented Programming concepts, to make our code more elegant.

This project intends to give you practice with basic Java concepts and basics of object-oriented software development and design, as well as some experience in developing software as requirements for a system's evolution. It also provides some good experience in building a system part by part from scratch, until an overall system is complete.

Preamble: "The Specifications"

The slides from this lesson module for the assignment act as "the specifications" for the system we are about to build.

"The specifications" in real-life software development is the document that details the features that should be implemented in any particular project. They are the features and requirements that you, the *Software Developer*, and the client have agreed should be implemented in the system.

As such, you should read these specifications carefully and ensure that your program implements the requirements of the specification correctly.

Tests will be run on your program to check that you have implemented these specifications correctly.

Note that for this project, we will provide **9 visible tests** that you can run to check that the basic functionality of your system is correct. However, there will also be **6 *hidden (i.e., invisible)* tests** that we will run when assessing the correctness of your code. You won't be able to see why your tests failed but you can see that they failed and rectify your code. So once your program passes the basic tests, *we strongly recommend that you perform further tests yourself*, to ensure that the required features have been implemented correctly.

How to read the specifications?

- Some parts of the specification may need you to use specific methods from Java in-built System or Util libraries. Or there can be certain assumptions that are okay to make by the students. We will provide additional help/suggestions using a green callout like the example provided below.

> ✅ **Tip:** use a specific string method like toLowerCase or something.

- Some specifications are treated as warnings. If not implemented correctly, they can cause incorrect output. They are shown in yellow warning callouts like the example provided below.

> ⚠️ **Warning:** Read this section carefully. Some additional texts to tell you what can go wrong.

- Some texts could be just informational. They neither cause you problems nor help you but guide you gently on what to do next. They will be shown in blue callouts like the example provided below.

> ℹ️ **Note:** Perhaps re-read this section first and then go to other section.

- Lastly, sometimes a developer thinks exhaustively and engages in continuous discussion with people to gather more information. We try to keep certain use cases out of scope as they bring

more complexity to the system. Some advanced developers can cope through it but we would like to keep some things simpler for beginners. Please read carefully through the **out-of-scope specifications** as well. This may be embedded as a red strip in the specifications sometimes.

> ❌ **Out of Scope Scenario:** This scenario .... is out of scope for this assignment

- Other than this, we will use the general informational, warning, error, and assumption callouts using blue, yellow, red, and green coloured callouts.
- Also, note that the code snippets have some characters in bold that represent the inputs to the program.

> ✔️ **Assumption:** Students can assume that test cases only contain outputs that are explicitly part of specifications. The test cases, visible or hidden, do not produce any output that is not mentioned in the specifications explicitly. Please note that the specifications will give you warnings and important notes where we expect special input handling. You should observe those and implement them accordingly as they will be tested while marking your program.

Preamble: Intended Learning Outcome

The Intended Learning Outcomes for this Assignment are mentioned below -

- Understand the process and methods of software design and implementation using Java programming language **including reading specifications**
- Apply the concepts of object-oriented design to the solution of computational problems **by using classes and objects**
- Read and understand a Java program of small to medium complexity **by reading and understanding the scaffold code presented to you and how to use the code or extend it**
- Write a Java program of small to medium complexity, which contains a number of classes with a console user interface - **by implementing the specifications**

The concepts used are from Week 1 to 3 and Constants from Week 4.

- DataTypes - char/bool/int/double/String etc.
- Branching - like if/else or switch cases to navigate the special use cases in specifications.
- Looping - use for/while/do-while loops for repeating certain actions
- Classes - identify the entities and encapsulate their data and actions together in a class.
- Constants - identify values that do not change and make them constants.

> ❌ Arrays are not allowed for this assignment. The intention is to test the above ILOs and hence use of Arrays/ArrayLists are prohibited.

> ⚠️ A warning for those that have previous experience programming in other programming languages (like C or Python) that follow a procedural programming paradigm: Be careful to develop your program using object oriented principles, as programming and structuring your code in a way that is not object-oriented is an easy way to lose marks for structure in the assignments.

Preamble: Java Coding Conventions

We will also be assessing your code for good structure and style.

Use methods when appropriate to simplify your code, and **avoid duplicate code.**

Use correct Java naming conventions for class names, variable names, and method names and put them in a well-organised way in your code i.e. it should be readable as well.

Make sure the **names** you choose are **meaningful**, to improve the readability of your code.

> **i** We will provide a marking scheme to help guide you in the development of your program. Also we will guide you how to model your program. But using correct syntaxes and conventions can be learnt here.

Academic Honesty

- All assessment items (assignments, tests and exams) must be your own, individual, original work.
- Any code that is submitted for assessment will be automatically compared against other students' code and other code sources using sophisticated similarity-checking software.
- Cases of potential copying or submitting code that is not your own may lead to a formal academic misconduct hearing.
- Potential penalties can include getting zero for the project, failing the subject, or even expulsion from the university in extreme cases.
- For further information, please see the university's Academic Honesty and Plagiarism website, or ask your lecturer.

You can find the Academic Integrity Module in your Canvas here.

PACMAN for Fun

Welcome to PACMAN for Fun.

We will not develop a UI-based game here as it needs an advanced level of programming. We will develop some features from a PACMAN Game with console-based input. Those who aren't aware of the PACMAN game can give it a little try here. The game has some key elements. There is a maze that has some boundaries and walls. The PACMAN can move around the maze and try to eat all the dots. There are some special food pellets that give them superpowers. There are 4 coloured monsters and the superpower helps to kill the monster. Otherwise, the monster can kill the PACMAN. When all the dots are eaten, the PACMAN game ends.

In this version, we will only be *setting up the layout of the maze for Pacman and moving them towards the special food*. In the next versions, we will enhance it further to resemble the real game where we can have monsters roaming around. The game ends when the PACMAN has eaten the special food. Not all the dots need to be eaten. There are no monsters in this assignment. There are various symbols we are going to use.

A maze looks like this

```
###########
#P--------#
#..-------#
#...------#
#.*..-----#
#.....----#
###########
```

The descriptions of the symbol are as below -

- # represents a boundary of the maze

- . represents a dot in the PACMAN game

- * represents the special food in the PACMAN game

- - represents a wall

- P represents PACMAN

How to create a maze and other details are provided in the slides later. To start with the program we will look at below things one by one -

- Main Menu

- Generating a Maze for PACMAN

- Move the PACMAN to eat the food and calculate the score for the game.

To start writing the program, there are certain files and empty methods that are provided to you. You should be able to run your Program by compiling it and running it from the EdStem platform.

> ⚠ **Warning**: Your program expects some command line parameters that we will discuss in **Maze section.** These command line parameters are set in EdStem platform and will be available to your program when you hit the submit/mark button. But if you are trying this in an IDE, you may have to set the command line parameters yourself using command line or IDE features. More details in Additional Help section.

Main Menu

Game Start

To start the game, your program must do two things -

1. Read some command line args to set the maze length and width.

2. Display a welcome message.

Your program must take command line inputs in the below order.

- maze length - Length of the maze
- maze width - Width of the maze
- food generator seed - some seed value to be passed to a Food Generator. The food Generator code has already been written for you. You need to invoke the constructor and methods appropriately.

More guidance for how to use these command-line params is provided for this here.

This means when you compile your program and run it, the commands on the terminal should look like this

```
javac GameEngine.java
java GameEngine 12 7 123
```

⚠️ **Warning:** You must not hard code this in your program. We may provide all kind of integer outputs to test these.

In case there are insufficient inputs (missing input) or if any of the input is 0 or negative, then exit the program by printing the error message -

```
javac GameEngine.java
java GameEngine 12 7 123
Invalid Inputs to set layout. Exiting the program now.
```

ℹ️ **Note:** These command line parameters are set in EdStem platform and will be available to your program when you hit the submit/mark button. But if you are trying this in an IDE, you may have to set the command line parameters yourself using command line or IDE features.

If the inputs are valid, the program will show a welcome message. A pre-defined method is provided to you already in the files. You need to invoke the method appropriately.

```
 ____        __          ___        _  _        __         __ _
(  _ \      / _\        / __)      ( \/ )      / _\       (  ( \
 ) __/     /    \      ( (__       / \/ \     /    \      /    /
(__)       \_/\_/       \___)      \_)(_/     \_/\_/      \_)__)

Let the fun begin
(`<     ...    ...   ...   ..........   ...
```

Main Menu

Once you have shown the display message, the next thing to show is the main menu for the user to select and perform some operations.

```
 ____        __          ___        _  _        __         __ _
(  _ \      / _\        / __)      ( \/ )      / _\       (  ( \
 ) __/     /    \      ( (__       / \/ \     /    \      /    /
(__)       \_/\_/       \___)      \_)(_/     \_/\_/      \_)__)

Let the fun begin
(`<     ...    ...   ...   ..........   ...

Select an option to get started.
Press 1 to select a pacman maze type.
Press 2 to play the game.
Press 3 to resume the game.
Press 4 to view the scores.
Press 5 to exit.
>
```

You should then take a user input and perform *one of the actions* out of the 5 given.

Exit Option

args. 长度

In case the user selects option 5, you must gracefully quit the program and print a goodbye message.

⚠ Warning: Do not use System.exit() as it is not a graceful exit.

Look at the complete output below.

ℹ Note: The text in bold represents user input here. You don't need to make the console inputs bold in your code.

```
 ____        __          ___        _  _        __         __ _
(  _ \      / _\        / __)      ( \/ )      / _\       (  ( \
```

```
 ) __/      /    \      ( (__       / \/ \      /    \      /    /
(__)        \_/\_/        \___)      \_)(_/      \_/\_/      \_)__)

Let the fun begin
(`<      ...     ...    ...   ..........    ...


Select an option to get started.
Press 1 to select a pacman maze type.
Press 2 to play the game.
Press 3 to resume the game.
Press 4 to view the scores.
Press 5 to exit.
> 5
Pacman says - Bye Bye Player.
```

Invalid Command

In case the user provides an invalid input like 7 or 9, the program should be able to print "Invalid Input" and show the main menu. Sample output

```
Select an option to get started.
Press 1 to select a pacman maze type.
Press 2 to play the game.
Press 3 to resume the game.
Press 4 to view the scores.
Press 5 to exit.
> 7
Invalid Input.
Select an option to get started.
Press 1 to select a pacman maze type.
Press 2 to play the game.
Press 3 to resume the game.
Press 4 to view the scores.
Press 5 to exit.
>
```
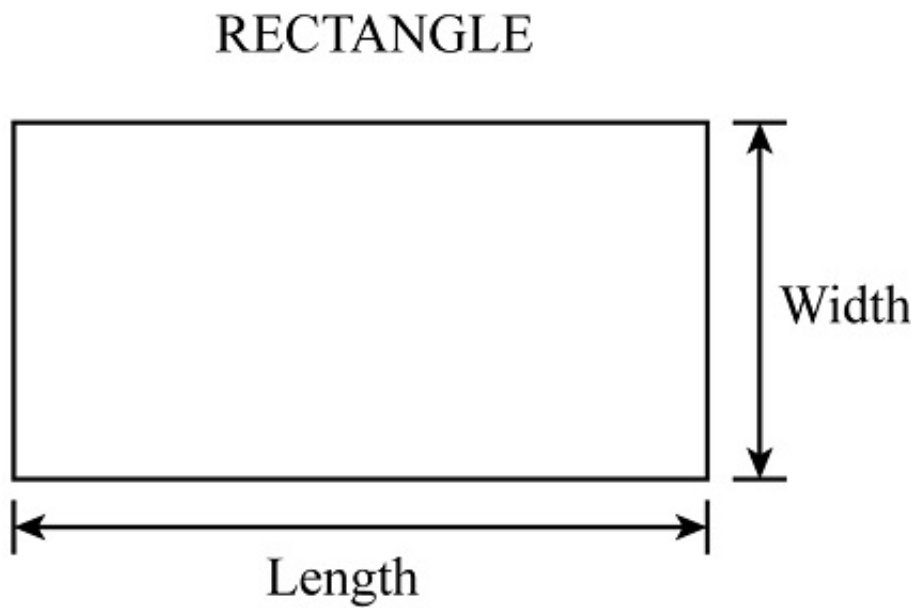
❌ **Out of Scope:** The program only expects an integer value for the main menu. You need not handle string or double values as input at this point of time.

Maze

Maze Types

There are three types of Maze we will explore in this assignment. A maze is rectangular in shape. The length and width are defined as follows.

RECTANGLE

Width

Length

Lower Triangular Maze

Here the upper triangle is blocked by the walls (-). The PACMAN can only navigate in the lower triangle area of the maze.

```
############
#P---------#
#..--------#
#...-------#
#.*..------#
#.....-----#
############
```

Upper Triangular Maze

Here the lower triangle is blocked by the walls (-). The PACMAN can only navigate in the upper triangle area of the maze.

```
############
#P.........#
#-.........#
#--...*....#
#---.......#
```

```
#----......#
###########
```

Horizontal Maze

In this maze, the PACMAN can navigate the maze using a horizontal path. Look at the dots as a path to move.

```
############
#P.........#
#.--------.#
#.....*....#
#.--------.#
#.........#
###########
```

How to build a maze?

You would be using Control Flows like branching and looping to build the maze. You have to use conditional logic to determine where the boundary/wall/PACMAN/dots go. Here are some guidelines for building the maze.

- Start by creating a maze as a rectangle.
- The number of columns is determined by maze Length.
- The number of rows is determined by maze width.
- The first row and last row and first column and last column are always boundaries shown by #.
- Next, you should consider creating the walls based on conditional logic and the maze type selected by the user.
- PACMAN game starts with the initial position of PACMAN in row = 1 and column = 1. (Remember, counting in JAVA starts from 0)
- For generating special food (*) you need to create a constructor of FoodGenerator and invoke appropriate methods. These methods would return you the row and column position of food.
- Lastly, you should fill the rest of the maze with dots.

Food Generator:

For your program to work correctly you must generate the food only in the constructor of Maze. This piece of code is to be written at a specific place and is highlighted in the scaffold presented to you in the comments.

> ℹ️ **Note**: How to create a triangular pattern? May be take a hint from these slides.

When the user selects option 1 from the main menu show them to choose a maze type. Once they have chosen a maze type, you must create a maze and show a message to the user.

Sample output for the above -

```
 ____          __           ___         _   _           __          __ _
(  _ \        / _\         / __)       ( \/ )         / _\        (  ( \
 ) __/       /    \       ( (__        / \/ \        /    \       /    /
(__)         \_/\_/        \___)       \_)(_/        \_/\_/       \_)__)

Let the fun begin
(`<     ...     ...    ...   .........   ...

Select an option to get started.
Press 1 to select a pacman maze type.
Press 2 to play the game.
Press 3 to resume the game.
Press 4 to view the scores.
Press 5 to exit.
> 1
Please select a maze type.
Press 1 to select lower triangle maze.
Press 2 to select upper triangle maze.
Press 3 to select horizontal maze.
> 2
Maze created. Proceed to play the game.
Select an option to get started.
Press 1 to select a pacman maze type.
Press 2 to play the game.
Press 3 to resume the game.
Press 4 to view the scores.
Press 5 to exit.
>
```

> ℹ️ **Note:** The text in bold represents user input here. You don't need to make the console inputs bold in your code.

Invalid Input

If the user inputs an invalid integer while selecting the maze type then the program should display an "Invalid Input" as a message and show the maze selection menu again.

```
Select an option to get started.
Press 1 to select a pacman maze type.
Press 2 to play the game.
Press 3 to resume the game.
Press 4 to view the scores.
Press 5 to exit.
> 1
Please select a maze type.
Press 1 to select lower triangle maze.
Press 2 to select upper triangle maze.
Press 3 to select horizontal maze.
> 6
```

```
Invalid Input.
Please select a maze type.
Press 1 to select lower triangle maze.
Press 2 to select upper triangle maze.
Press 3 to select horizontal maze.
> 3
Maze created. Proceed to play the game.
Select an option to get started.
Press 1 to select a pacman maze type.
Press 2 to play the game.
Press 3 to resume the game.
Press 4 to view the scores.
Press 5 to exit.
>
```

Play the Game

Option 2: Start the Game

When the user selects option 2 to start the game, there are three possibilities-

1. You haven't selected a maze type.
2. You have selected a maze type and are ready to play the game.
3. You have selected a maze type but you are in the middle of a game that is not over yet.

Scenario 1: Maze Type not selected

In this case, you should show an error message and repeat the main menu.

```
Select an option to get started.
Press 1 to select a pacman maze type.
Press 2 to play the game.
Press 3 to resume the game.
Press 4 to view the scores.
Press 5 to exit.
> 2
Maze not created. Select option 1 from main menu.
Select an option to get started.
Press 1 to select a pacman maze type.
Press 2 to play the game.
Press 3 to resume the game.
Press 4 to view the scores.
Press 5 to exit.
>
```

Scenario 2: Maze Type selected new game started

If the maze is created you can start the game. At the start of the new game, you show some information about how you will gain points and show the user a menu to move the PACMAN. How to move the PACMAN will be shown in the move menu here.

```
Select an option to get started.
Press 1 to select a pacman maze type.
Press 2 to play the game.
Press 3 to resume the game.
Press 4 to view the scores.
Press 5 to exit.
> 2
Move the Pacman towards the food pellet.
  > You gain 20 points when Pacman get the food.
  > You lose 0.5 point when you hit the wall/boundary.
  > Score = 20 * Food - 0.5 * hits - 0.25 * moves.
```

```
###########
#P--------#
#..-------#
#...------#
#.*..-----#
#.....----#
###########
Press W to move up.
Press A to move left.
Press S to move down.
Press D to move right.
Press Q to exit
>
```

Scenario 3: Maze Type selected but previous game in progress

In this case, the program will show a message to the user that the previous game is in progress and ask if the user wants to keep the existing game.

- If yes, the user presses N, then the program prompts the main menu again.
- If not, the user presses any other key, then the program will discard the current game and start over a new game.

Sample output here -

Output for when the user selects N

```
Select an option to get started.
Press 1 to select a pacman maze type.
Press 2 to play the game.
Press 3 to resume the game.
Press 4 to view the scores.
Press 5 to exit.
> 2
Previous game hasn't ended yet. Do you want to discard previous game?
Press N to go back to main menu to resume the game or else press any key to discard
> N
Select an option to get started.
Press 1 to select a pacman maze type.
Press 2 to play the game.
Press 3 to resume the game.
Press 4 to view the scores.
Press 5 to exit.
```

Output when the user selects any other key

```
Select an option to get started.
Press 1 to select a pacman maze type.
Press 2 to play the game.
```

```
Press 3 to resume the game.
Press 4 to view the scores.
Press 5 to exit.
> 2
Previous game hasn't ended yet. Do you want to discard previous game?
Press N to go back to main menu to resume the game or else press any key to discard.
> S
Move the Pacman towards the food pellet.
  > You gain 20 points when Pacman get the food.
  > You lose 0.5 point when you hit the wall/boundary.
  > Score = 20 * Food - 0.5 * hits - 0.25 * moves.
###########
#P--------#
#..-------#
#...------#
#.*..-----#
#.....----#
###########
Press W to move up.
Press A to move left.
Press S to move down.
Press D to move right.
Press Q to exit
>
```

Option 3: Resume the Game

A game is paused when the PACMAN hasn't reached/eaten the special food and you quit moving the PACMAN to go back to the main menu. More information here. You can only resume a game that is ongoing. If you do not have any ongoing game then you should show an error message. There are three scenarios here -

1. You haven't selected a maze type
2. You are resuming a paused game.
3. The previous game has ended and the new game hasn't started.

Scenario 1: Maze Type not selected.

```
Select an option to get started.
Press 1 to select a pacman maze type.
Press 2 to play the game.
Press 3 to resume the game.
Press 4 to view the scores.
Press 5 to exit.
> 3
Maze not created. Select option 1 from main menu.
Select an option to get started.
Press 1 to select a pacman maze type.
Press 2 to play the game.
Press 3 to resume the game.
Press 4 to view the scores.
```

```
Press 5 to exit.
>
```

Scenario 2: Resuming a paused game

```
Select an option to get started.
Press 1 to select a pacman maze type.
Press 2 to play the game.
Press 3 to resume the game.
Press 4 to view the scores.
Press 5 to exit.
> 3
Restart your game from the last position you saved.
###########
#.--------#
#.P-------#
#...------#
#.*..-----#
#.....----#
###########
Press W to move up.
Press A to move left.
Press S to move down.
Press D to move right.
Press Q to exit
>
```

Scenario 3: No paused game found

```
Select an option to get started.
Press 1 to select a pacman maze type.
Press 2 to play the game.
Press 3 to resume the game.
Press 4 to view the scores.
Press 5 to exit.
> 3
No paused game found. Select option 2 from main menu to start a new game.
Select an option to get started.
Press 1 to select a pacman maze type.
Press 2 to play the game.
Press 3 to resume the game.
Press 4 to view the scores.
Press 5 to exit.
>
```

Option 4: View Score

There are multiple possible scenarios.

1. No completed games found - Show an error message.

2. All games have been completed - Show scores of all games.
3. Some games have been completed and one game is ongoing - Only show scores of all completed games.

How do you generate # Game i.e., no of games played? Every time you start a new game, you should increment the game counter. Even if the game is discarded, the game counter increases with the next new game. In the below output, the score for game 3 is not shown as it was discarded.

> **i** **Note:** You should maintain a variable for game counter in such a way that every time game ends, the value of game counter is not lost.

Game Scores are to be shown in a formatted way.

- # Game means game number
- # Hits means the number of times Pacman hit the boundary and the wall
- # Moves means the number of moves made by PACMAN to reach the special food. **A hit is not included as a move.**
- # Score means score for the game calculated as 20 * Food - 0.5 * number of hits - 0.25 * number of moves.

Here is a sample output of game scores.

```
|  # Game|  # Hits| # Moves| # Score|
|========|========|========|========|
|       1|       0|       3|   19.25|
|       2|       1|       7|   17.75|
|       4|       1|       7|   17.75|
```

> **✓** **Tip:** Use String.format method to format the strings. Use the String formatter **"|%8s|%8s|%8s|%8s|%n"** for the header and formatter **"|%8d|%8d|%8d|%8.2f|%n"** for printing the actual values.

Scenario 1: No completed games found

```
Select an option to get started.
Press 1 to select a pacman maze type.
Press 2 to play the game.
Press 3 to resume the game.
Press 4 to view the scores.
Press 5 to exit.
> 4
No completed games found.
Select an option to get started.
Press 1 to select a pacman maze type.
Press 2 to play the game.
Press 3 to resume the game.
```

```
Press 4 to view the scores.
Press 5 to exit.
>
```

Scenario 2 & 3: Show all completed games score

```
Select an option to get started.
Press 1 to select a pacman maze type.
Press 2 to play the game.
Press 3 to resume the game.
Press 4 to view the scores.
Press 5 to exit.
> 4
|  # Game|  # Hits| # Moves| # Score|
|========|========|========|========|
|       1|       0|       3|   19.25|
|       2|       1|       7|   17.75|
Select an option to get started.
Press 1 to select a pacman maze type.
Press 2 to play the game.
Press 3 to resume the game.
Press 4 to view the scores.
Press 5 to exit.
>
```

❌ **Out of Scope:** When your program terminates you don't have to save the game counter or any other data. The program resets to default.

Move Pacman

When the user selects option 2 or 3 and plays the game, you should show the move menu. However, the message for starting a game shows information on how to gain points (See slides here and section Start the Game> Scenario 2). If the game is resumed after pausing, the message will be different (See the slides here and section Resume the Game> scenario 2). For simplification, we are only showing the Move menu for a new game in the below outputs.

```
Select an option to get started.
Press 1 to select a pacman maze type.
Press 2 to play the game.
Press 3 to resume the game.
Press 4 to view the scores.
Press 5 to exit.
> 2
Move the Pacman towards the food pellet.
  > You gain 20 points when Pacman get the food.
  > You lose 0.5 point when you hit the wall/boundary.
  > Score = 20 * Food - 0.5 * hits - 0.25 * moves.
###########
#P--------#
#..-------#
#...------#
#.*..-----#
#.....----#
###########
Press W to move up.
Press A to move left.
Press S to move down.
Press D to move right.
Press Q to exit.
>
```

⚠️ **Warning: in this case you will expect a String input - w/a/s/d/q. This can be upper or lower case. You must handle the input accordingly.**

Invalid Input

If the user enters any other string like "hi", you must prompt "Invalid input". Sample output below -

```
Press W to move up.
Press A to move left.
Press S to move down.
Press D to move right.
Press Q to exit.
> hi
Invalid Input.
```

```
###########
#P--------#
#..-------#
#...------#
#.*..-----#
#.....----#
###########
Press W to move up.
Press A to move left.
Press S to move down.
Press D to move right.
Press Q to exit.
>
```

Movement - Right/Left/Top/Bottom

In case the user wants to move in any direction, the position of PACMAN should change. When the user selects option 2 for the first time, by default the top left corner is the starting position of PACMAN. If the user resumes a game, then you should start from the last saved position of PACMAN.

Every time PACMAN makes a valid move you should count the number of moves, and every time they hit a wall or a boundary you should count the no of hits. A hit is not counted as a move.

Move Right

```
###########
#.--------#
#P.-------#
#...------#
#.*..-----#
#.....----#
###########
Press W to move up.
Press A to move left.
Press S to move down.
Press D to move right.
Press Q to exit.
> D
###########
#.--------#
#.P-------#
#...------#
#.*..-----#
#.....----#
###########
Press W to move up.
Press A to move left.
Press S to move down.
Press D to move right.
Press Q to exit.
>
```

Move Left

```
###########
#.--------#
#.P-------#
#...------#
#.*..-----#
#.....----#
###########
Press W to move up.
Press A to move left.
Press S to move down.
Press D to move right.
Press Q to exit.
> A
###########
#.--------#
#P.-------#
#...------#
#.*..-----#
#.....----#
###########
Press W to move up.
Press A to move left.
Press S to move down.
Press D to move right.
Press Q to exit.
>
```

Move Down

```
###########
#P--------#
#..-------#
#...------#
#.*..-----#
#.....----#
###########
Press W to move up.
Press A to move left.
Press S to move down.
Press D to move right.
Press Q to exit.
> S
###########
#.--------#
#P.-------#
#...------#
#.*..-----#
#.....----#
###########
Press W to move up.
Press A to move left.
```

```
Press S to move down.
Press D to move right.
Press Q to exit.
>
```

Move up

```
###########
#.--------#
#P.-------#
#...------#
#.*..-----#
#.....----#
###########
Press W to move up.
Press A to move left.
Press S to move down.
Press D to move right.
Press Q to exit.
> W
###########
#P--------#
#..-------#
#...------#
#.*..-----#
#.....----#
###########
Press W to move up.
Press A to move left.
Press S to move down.
Press D to move right.
Press Q to exit.
>
```

Invalid Moves

If PACMAN is already at the leftmost position/rightmost position/ topmost position or bottommost position and the user wants to move left/right/up/down respectively, that would be an invalid move and PACMAN may hit a boundary or a wall. The program should print appropriate messages for the invalid move. Look at the sample outputs below -

Hit a wall

```
###########
#P--------#
#..-------#
#...------#
#.*..-----#
#.....----#
###########
Press W to move up.
```

```
Press A to move left.
Press S to move down.
Press D to move right.
Press Q to exit.
> D
You have hit a wall.
###########
#P--------#
#..-------#
#...------#
#.*..-----#
#.....----#
###########
Press W to move up.
Press A to move left.
Press S to move down.
Press D to move right.
Press Q to exit.
>
```

Hit a boundary

```
###########
#P--------#
#..-------#
#...------#
#.*..-----#
#.....----#
###########
Press W to move up.
Press A to move left.
Press S to move down.
Press D to move right.
Press Q to exit.
> A
You have hit the boundary.
###########
#P--------#
#..-------#
#...------#
#.*..-----#
#.....----#
###########
Press W to move up.
Press A to move left.
Press S to move down.
Press D to move right.
Press Q to exit.
>
```

Exiting from the PACMAN Movement Menu

If the user selects the option Q then the program should save the current position of PACMAN and go back to the main menu.

```
###########
#P--------#
#..-------#
#...------#
#.*..-----#
#.....----#
###########
Press W to move up.
Press A to move left.
Press S to move down.
Press D to move right.
Press Q to exit.
> S
###########
#.--------#
#P.-------#
#...------#
#.*..-----#
#.....----#
###########
Press W to move up.
Press A to move left.
Press S to move down.
Press D to move right.
Press Q to exit.
> S
###########
#.--------#
#..-------#
#P..------#
#.*..-----#
#.....----#
###########
Press W to move up.
Press A to move left.
Press S to move down.
Press D to move right.
Press Q to exit.
> D
###########
#.--------#
#..-------#
#.P.------#
#.*..-----#
#.....----#
###########
Press W to move up.
Press A to move left.
Press S to move down.
Press D to move right.
```

```
Press Q to exit.
> Q
Your game is paused and saved.
Select an option to get started.
Press 1 to select a pacman maze type.
Press 2 to play the game.
Press 3 to resume the game.
Press 4 to view the scores.
Press 5 to exit.
>
```

If the user resumes the game by selecting option 3 from the main menu, the game must restart at the same location.

```
############
#.---------#
#..--------#
#.P.-------#
#.*..------#
#.....-----#
############
Press W to move up.
Press A to move left.
Press S to move down.
Press D to move right.
Press Q to exit.
> Q
Your game is paused and saved.
Select an option to get started.
Press 1 to select a pacman maze type.
Press 2 to play the game.
Press 3 to resume the game.
Press 4 to view the scores.
Press 5 to exit.
> 3
Restart your game from the last position you saved.
############
#.---------#
#..--------#
#.P.-------#
#.*..------#
#.....-----#
############
Press W to move up.
Press A to move left.
Press S to move down.
Press D to move right.
Press Q to exit.
>
```

Ending a Game

When the PACMAN has reached the special food, you must end the game by showing the score of the current game. This should also be visible if the user selects option 4 to see the scores of the completed games. See the output below -

```
Select an option to get started.
Press 1 to select a pacman maze type.
Press 2 to play the game.
Press 3 to resume the game.
Press 4 to view the scores.
Press 5 to exit.
> 3
Restart your game from the last position you saved.
###########
#.---------#
#..--------#
#.P.-------#
#.*..------#
#.....-----#
###########
Press W to move up.
Press A to move left.
Press S to move down.
Press D to move right.
Press Q to exit.
> S
###########
#.---------#
#..--------#
#...-------#
#.P..------#
#.....-----#
###########
Game has ended! Your score for this game is 14.25
Select an option to get started.
Press 1 to select a pacman maze type.
Press 2 to play the game.
Press 3 to resume the game.
Press 4 to view the scores.
Press 5 to exit.
> 4
|  # Game|  # Hits| # Moves| # Score|
|========|========|========|========|
|       1|       8|       7|   14.25|
Select an option to get started.
Press 1 to select a pacman maze type.
Press 2 to play the game.
Press 3 to resume the game.
Press 4 to view the scores.
Press 5 to exit.
>
```

That would be all for now. We will come back with more features for PACMAN with

monsters in the next version. Happy Coding Everyone!

PACMAN for Fun

Some starter code has been provided for you. Write the code to implement your program there

We also provided some tests for your code, which will run automatically every time you hit "Mark". Your submission is saved when you hit the Mark button. You can make as many submissions as you want and run these tests as many times as you would like until the submission deadline.

> ⚠️ **Submission will close on 28 Aug 2024 5pm AEST.**

Happy coding!

> ⚠️ **Warning! You must make sure that your code runs on edstem. If your code does not compile on edstem, 0 marks will be given.**

COMP90041 Assignment 1: Marking Scheme

Program Presentation

Including layout and style, readability, adherence to coding expectations and conventions, general care and appearance. The full mark for this section of marking is **6**.

Marks awarded

Some subset of the following lines will be selected by the marker.

Gain **0.5** marks for any of the issues listed below.

- All choices of variable names were meaningful;
- All choices of method names were meaningful;
- variable and method names follow Java convention (camelCase);
- constant names follow Java convention (UPPER_SNAKE_CASE);
- class names follow Java convention (UpperCamelCase);
- comments were sufficient and meaningful;
- consistent bracket placement;
- indentation was consistent;
- whitespace was used to separate logical blocks of code;
- authorship statement (name, university email, student number) provided;
- all magic numbers (essentially numbers other than 0 or 1) were assigned to constants;
- switch case labels were assigned to constants

Deductions

- stylistic issue, if major **-1.0** mark per issue; if minor, **-0.5** mark per issue

Structure and Approach

Including decomposition into methods, declaration of instance variables at the appropriate locations, and choice of parameters to methods.

The full mark for this section of marking is **9**.

Marks awarded

Some subset of the following lines will be selected by the marker.

Gain **1** mark for any issue listed below.

- Good use of methods;
- No methods were too long or too complex;
- code was not duplicated (tasks to be done in more than one place are in the method);
- simple algorithmic approach to solving the problem;
- clearly structured code;
- no more than 3 `static` methods (including main) were used -- most should be bound to objects;
- no more than 4 `static` variables were used -- most should be non-static ("`final static`" constants are not variables);
- 'appropriate use of encapsulation; limiting the scope of variables to `private` except where reasonable justification is given.
- Code File Organisation - instance variables, constructors, and methods are added in a class in the right order.

Deductions

- structural issue, if major **-2.0** marks per issue, otherwise **-1.0** mark per issue.
- ❌ **If you use arrays, we will deduct 3 marks for that.**

Program Execution

Including compilation, execution of test data, output presentation and readability.

Programs that do not compile in the test environment will lose all marks in this section. Be sure to verify your submission and **check the output** before you say "finished" to yourself.

The full mark for this section of marking is **15**.

Marks awarded

- Gain **1** marks per test case passed;
- Gain **0.5** marks for tests with *slightly* different output (e.g., small changes in whitespace; if the marker says the difference is not slight, that is final. This should not occur if the available test cases are all checked.);
- Gain **0** for other failed tests

Visible tests passed: / 9

Hidden tests passed: / 6

Total tests passed: / 15

Total Marks for the Assignment: 30

Total Marks for the Assignment(scaled) : 30/2 = 15

Assessment

This project is worth **15%** of the total marks for the subject. Your Java program will be assessed based on the correctness of the output as well as the quality of code implementation.

Automatic tests will be conducted on your program by compiling, running, and comparing your outputs for several test cases with generated expected outputs. The automatic test will deem your output wrong if your output does not match the expected output, even if the difference is just having extra space or missing a colon. Therefore, it is crucial that your output follows exactly the same format shown in the provided examples.

> ⚠ Passing the tests we provide here on edstem does not mean you will score full marks. Each submission will further be subject to hidden tests as well as manual inspection by our teaching team.

The syntax **import** is available for you to use standard java packages. However, DO NOT use the package syntax to customize your source files. The automatic test system may not deal with customized packages. If you are using Netbeans, IntelliJ or Eclipse for your development, be aware that the project name may automatically be used as the package name. You must remove lines like:

```
package Assignment1;
```

at the beginning of the source files when you copy them to edstem. Otherwise, the automatic tests may fail and tell you so.

Submission

Your submission should have at least two Java source code files. You must name them **GameEngine.java, Maze.java.**

Starter code has been provided to you here on the edstem platform, but you are welcome to develop the project outside of the edstem environment, using your own IDEs.

Submission is made via edstem, however, and will be whatever code is saved on the Code Challenge when the project deadline closes.

> ⚠️ Your code MUST compile and run here on edstem. If your code does not compile we cannot mark it and you risk getting 0 marks.

"I can't get my code to work on edstem but it worked on my local machine" is <u>not</u> an acceptable excuse for late submissions. In addition, **submissions via email will not be accepted.**

Be sure to copy your code into your Code Challenge workspace well before the deadline to avoid being locked out of submissions last minute.

Only the last version of your code before the submission deadline will be graded. It is highly recommended that you update your code on edstem frequently and well before the submission deadline. Last-minute *"connection errors"* are not a valid excuse.

In case you need an extension due to valid reasons up to 3 days, please fill out the form mentioned in the slides here and send it to the subject coordinator's email. We will follow up with you. Ensure you have a valid reason and proper documentation with you before seeking the extension. If you seek an extension beyond 3 days, check the new Special Consideration Module on Canvas and raise your request using this portal. See more here.

YOU'VE GOT THIS!

memegenerator.net
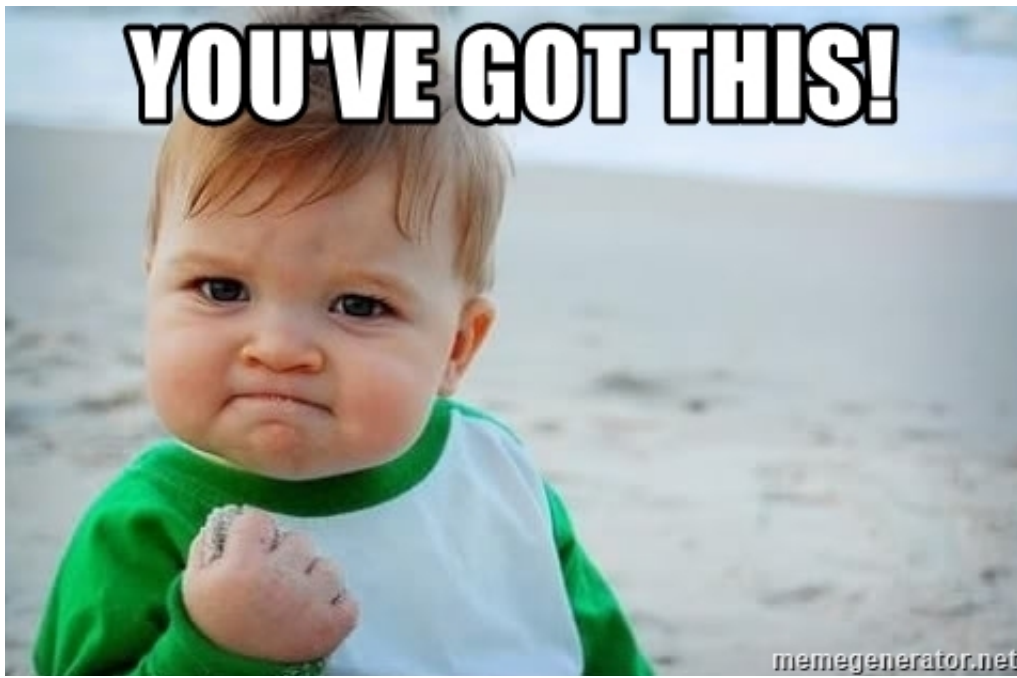
FAQ

1. Do I need to make my inputs bold?
   - No, the bold is to highlight that they are inputs to the program, you should not hard code them in your program.
2. Every file has an Author, Student Id and Email. Do we need to write that down in every file?
   - Yes, please fill authorship statement in every file including the new files that you create. Otherwise, you may lose marks. See the Marking Scheme.
3. Can I create new files or should I only work with the existing files?
   - Yes, you can create new files as well.
4. Do we need to handle all kinds of invalid inputs? Will this be tested?
   - We have mentioned in the Main menu that we only expect integers. You do not need to handle String/Double or other kinds. For the Movement Menu, the inputs are strings W/A/S/D and not characters. We may provide an invalid input like "hi". So handle them as strings.
5. I am getting "NoSuchElementException" and my program terminates. What do I do?
   - There are two things you should do -
     - Ensure you are using only one Scanner throughout all the files. This scanner object is provided to you in the GameEngine file. You can use this Scanner in other files by writing `GameEngine.keyboard.nextLine()`. This error occurs when you try to use multiple Scanner and they are closed as they go out of scope.
     - Ensure that you are using nextLine method instead of using nextInt or nextDouble. nextInt, nextDouble won't read your new lines (\n) and your program will read the inputs incorrectly leading to exceptions. Use nextLine for reading inputs always and you can convert them to int/double/ etc by using Integer.parseInt() or Double.parseDouble() methods. See the slides here.
6. How do I save my game score or pause the game and restart at the same location?
   - The object contains references. As long as you keep using the same object reference, your data is stored in that object. If you pass your object to a method, the method has access to all the data. For more reference, rewatch the Week 4 Lecture slides and video.
7. My test cases fail but I cannot see a difference. What's wrong?
   - If you cannot see a difference in text, it means you may be missing a space or new line.
8. What does this mean in my test cases? It fails due to "↵" character in red.
   - It means you are missing a new line.
9. The output in the specifications is shown as

```
Please select a maze type.
Press 1 to select lower triangle maze.
Press 2 to select upper triangle maze.
Press 3 to select horizontal maze.
> 2
Maze created. Proceed to play the game.
```

However my test results show output difference. What's the problem?

```
Please select a maze type.
Press 1 to select lower triangle maze.
Press 2 to select upper triangle maze.
Press 3 to select horizontal maze.
> Maze created. Proceed to play the game.
```

- Well, your test outputs don't show the inputs however your program should generate output with appropriate new lines as per the specifications. You should look for other differences in spelling or new line issues.

We will add more FAQs as we receive more student queries. See you later, alligator!