Mock05: GUI

1. Instruction

a) Create a project named "**Q5**" inside your CP_seatNo_{ID}_{firstname} folder in c:\temp. **If folder Q5 already exists, you must delete it first (Otherwise your file may get corrupted!)**

b) Use JavaFX library. --module-path "path to JavaFX lib folder" --add-modules javafx.controls, javafx.fxml, javafx.graphics, javafx.media

c) Copy "res" folder from "Q5" into your project root folder.

d) Copy all folders inside "src" of into the project's "src" folder. Make sure "res" is a resource folder.

e) Please see the implementation details in the next section.

Export jar (**q5.jar**) file that **includes source code, resources (image and maps), and .class files** and put it in **root** folder of your project. Your jar file must have source code and must be runnable.

<u>YOU MUST EXPORT JAR FILE containing all mentioned contents. IF YOUR FILE DOES NOT CONTAIN EVEN 1 MENTIONED FILE, YOUR CODE WILL NOT BE MARKED</u>.

2. Problem Statement: Nokotan Pathfinder

In this problem, you will implement some GUIs in a puzzle game “Nokotan Pathfinder”.

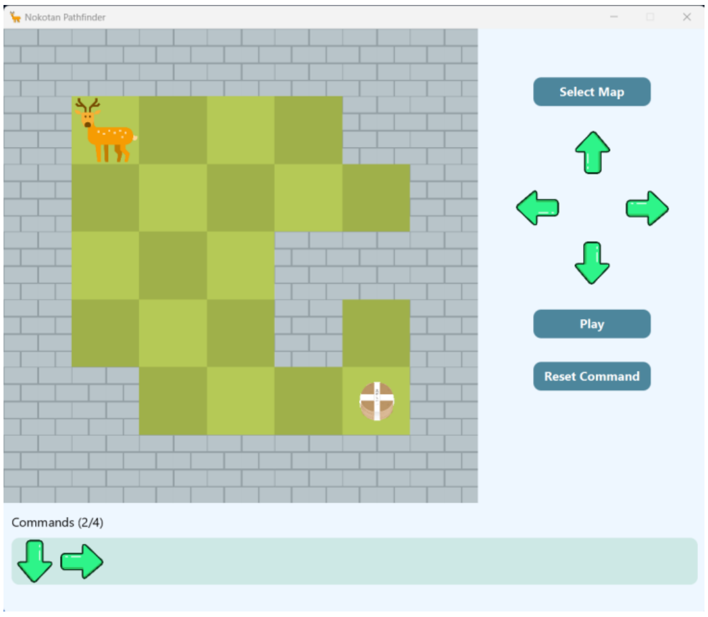


Figure 1: Overview of Nokotan Pathfinder

“Nokotan Pathfinder” is a puzzle game inspired from “Ice Sliding Puzzle”, and “My Deer Friend Nokotan”, a Japanese animation. The rules of the game are as follows:

- The goal of this game is to **help a deer to stand** **<u>exactly</u>** **on a deer cracker**. The **deer can only dash or slide** in 1 direction until it collides with a wall.

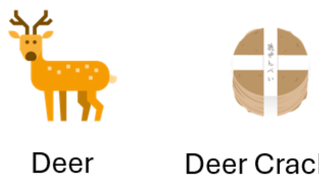
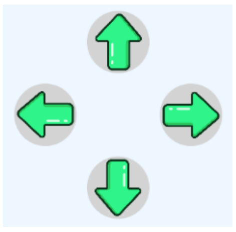


Figure 2: Deer and Deer Cracker

- To control the deer, **you must fill your commands (up, down, left, right)** in the CommandController pane. 

The selected commands will be shown at the bottom of the screen. For example, if the input commands are “down right down right”, the bottom of the screen will show the command in Figure 3 (left). The deer will move as shown in Figure 3 (right).

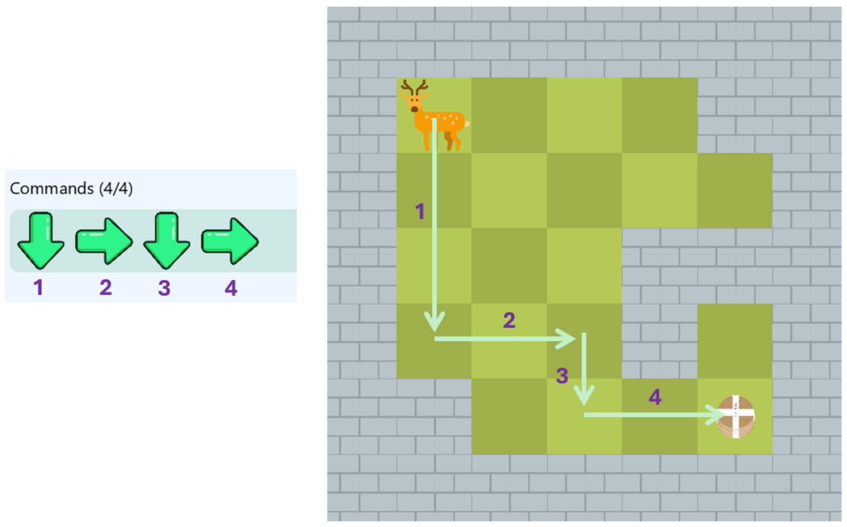


Figure 3: Example of how commands work

- Take note that **there are maximum commands that can be used** for each map.

The application consists of 3 main panes, which are: game pane, control pane, and command pane. The game pane includes a grid of a map. The control pane includes a map selector, arrows which can be pressed to add command, play button, and reset button. The command pane shows commands which are input.

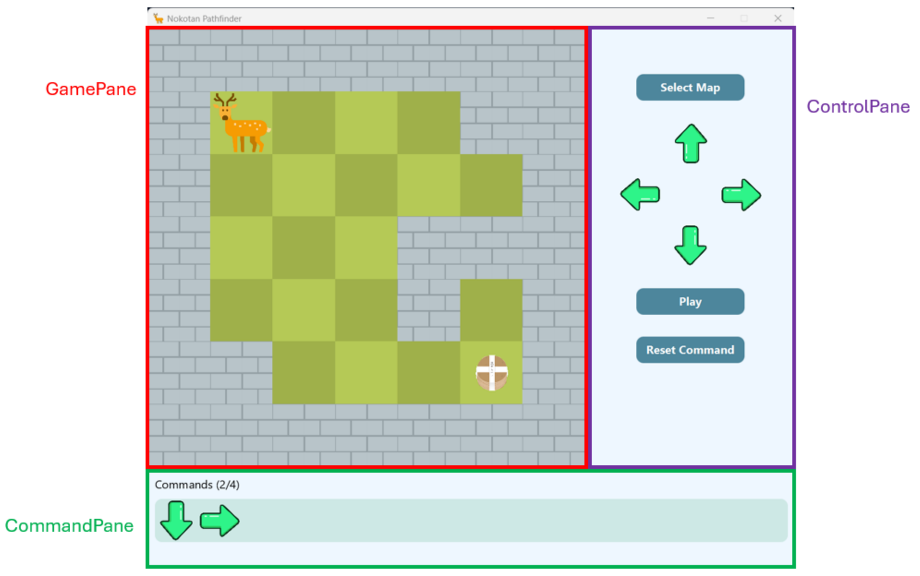
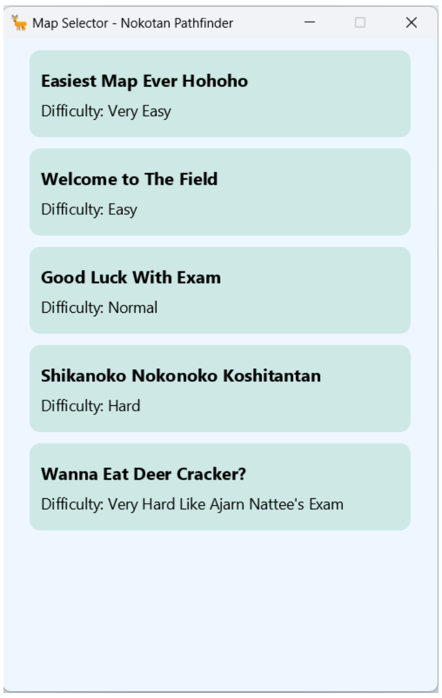


Figure 4: Main panes in the application

To play a game, the player must select a map by pressing "Select Map" button. Then, a map selector window (another stage) will be displayed. Select a map to start the game.


Map Selector - Nokotan Pathfinder

Easiest Map Ever Hohoho
Difficulty: Very Easy

Welcome to The Field
Difficulty: Easy

Good Luck With Exam
Difficulty: Normal

Shikanoko Nokonoko Koshitantan
Difficulty: Hard

Wanna Eat Deer Cracker?
Difficulty: Very Hard Like Ajarn Nattee's Exam

Figure 5: Map Selector Window (Stage)

Then, the map will be generated. The player adds commands by pressing arrows at the CommandController pane. After the player filled commands, the "Play" button should be pressed to let the deer walk. If the deer stands exactly on deer cracker after all commands are executed, the game will end and "**You Win!**" message will be shown at the top-right corner of the application. If the deer fails to stand on deer cracker, the game would show "**Git Gud**" message instead.

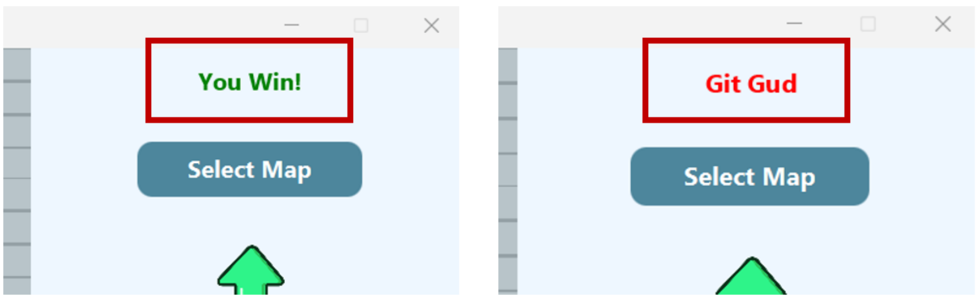


Figure 6: Message shown at the top-right corner of application

After the player wins, the player can play again by choosing a new map. Also, the player can reset filled command by pressing “Reset Command” button. A demo.mp4 file shows a video of how this game works.

3. Implementation Details

You will only need to implement a few methods. The location you must write/fix the code will be marked in this color and will be labeled with “TODO” in the code. However, it is recommended that you read the descriptions of other classes as you may need them.

3.1 package application

This package is the entry point of this application.

3.1.1 class Main extends Application

This class is the entry point of this JavaFX application.

3.2 package gui

This package contains some small UI components for the application.

3.2.1 class ArrowPane extends Pane

The is the pane which will be shown in the CommandPane. It will show how a command looks like (up, down, left, right).

Field

Field	Description
- Command command	The command in this arrow pane (up, down, left, right)

Method

Method	Description
+ ArrowPane(Command command)	Constructor which initializes new ArrowPane with specific command.

- void setImageBackground(Image image)	Set background of the pane
- void setImage()	Set image background depends on command in object
# Command getCommand()	Return command in the object

3.2.2 class CellPane extends Pane

This is the pane that will show in the game pane, each cell will show an image that indicates an object in the map.

Field

Field	Description
- int row	The row of this cell in the map
- int col	The column of this cell in the map

Method

Method	Description
+ CellPane(double width, double height, int row, int col)	Constructor which initializes new CellPane with specific width, height, row, and column.
+ void setImage()	Set background of the cell
+ Tile getState()	Get tile state of the cell
Other getters and setters	

3.2.3 class CommandArrowPane extends StackPane

This is the arrow button pane that user can click to add commands. This will be in CommandControllerPane. /* You must implement a method in this class */

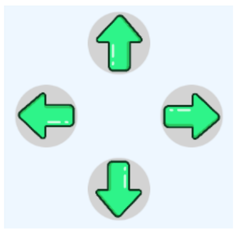
Method

Method	Description
+ CommandArrowPane(Command command)	Constructor which initializes new CommandArrowPane with specific command. /* TODO: Complete the code */ When player click on this pane, add command by calling GameSystem.*getInstance*().addCommand(command).

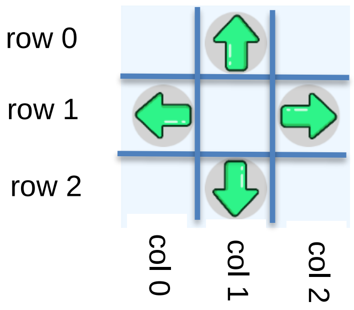
3.2.4 class CommandControllerPane extends GridPane

This is a grid pane that contains 4 arrow buttons to add commands to the current command list. The grid is 3x3 which is in ControlPane. /* You must implement a method in this class */

The grid looks like this:



The rows and columns numbers are:



Method

Method	Description

+ CommandControllerPane()	Constructor. /* TODO: Complete the code */ Just add upPane, leftPane, rightPane, and downPane to the grid correctly.

3.2.5 class CommandPane extends VBox

This pane is at the bottom of the application, which contains current commands during the game.

Field

Field	Description
<u>- CommandPane instance</u>	The singleton instance of CommandPane
- Text counterText	The text that shows title of CommandPane, including number of current commands and maximum commands.
- HBox commandContainer	The box that contains current commands

Method

Method	Description
- CommandPane()	Constructor which initializes a new CommandPane.
- void initializeText()	Initialize the counter text
- void initializeCommandContainer()	Initialize the command container
+ void updateCommandPane()	Update the command pane by doing following steps: ● Remove all children in the command container.

	● Get all the current commands. Then, for each command from the current commands, add a corresponding arrow pane (ArrowPane) to the commandContainer. ● Set the text of counter text to "Commands (***<number_of_current_commands>/<maximum_commands>***)" For example, if there are 5 current commands and the map limit maximum command at 8, then the text must show "Commands (5/8)"
<u>+ CommandPane getInstance()</u>	Return the singleton instance of CommandPane.

3.2.6 class ControlPane extends VBox

This pane is at the right of the application, which contains buttons for controlling the game and button for selecting map.

Field

Field	**Description**
<u>- ControlPane instance</u>	The singleton instance of ControlPane
- Button selectMapButton	The map selection button
- CommandControllerPane commandArrowPane	The arrow pad to let player add commands
- Button playButton	The button to execute commands
- Button resetButton	The button to remove all current commands
- Text notificationText	The text which shows after command execution

Method

Method	Description
- ControlPane()	Initialize new ControlPane
- void initializeNotificationText()	Initialize notification text which is hidden
- void showNotification(String text, Color colorText)	Show notification text with specific text and color
- void hideNotification()	Hide the notification text
- void initializePlayButton()	Initialize play button
- void initializeSelectMapButton()	Initialize select map button
- void initializeResetButton()	Initialize reset button
- void initializeCommandArrowPane()	Initilaize command arrow pane
+ ControlPane getInstance()	Return the singleton instance of ControlPane

3.2.7 class GamePane extends GridPane

This pane is at the top-left of the application, which contains GUI for the map.

/* You must implement some methods in this class */

Field

Field	Description
- GamePane instance	The singleton instance of GamePane
- List<List<CellPane>> gridCellPane	The 2-dimensional list containing cell pane for the map. (This is created because there is no API/method for accessing to child of GridPane directly.)
- double tileSize	The width and height of each cell pane

Method

Method	Description
- GamePane()	Initialize new GamePane
<u>+ GamePane getInstance()</u>	Return the singleton instance of GamePane

+ void initTiles()	/* TODO: Complete the remaining code */
	Reset the map and initialize all tiles again.
	● Remove all children from the grid pane. ***(Already Provided)***
	● Calculate each tile size. ***(Already Provided)***
	● Initialize 2-dimensional list for gridCellPane. ***(Already Provided)***
	● Add new cell panes with width and height equal to tileSize into each grid slot. The size of the grid is the width and the height of the map.
	● Also, add each created cell pane into gridCellPane.
	
	Hint 1: Observe a class **Map** to retrieve information of the map (e.g. map width, map height, ...)
	Hint 2: To add objects into 2-dimensional list, first, you must add 1-dimensional arraylist for each new row. For example, if the list is a 2-dimensional list, then:
	
	`for (int i = 0; i < n; i++) {`
	`    list.add(new ArrayList<CellPane>());`
	`    ...`
	`}`

3.3 package gui.selector

This package contains some UI for the map selector stage for the application.

3.3.1 class MapButtonPane extends VBox

This button is in MapSelectorPane, which shows the name and difficulty of the map. When the player clicks on the button, the game will generate map tiles and close the map selection stage.

3.3.2 class MapSelectorPane extends VBox

This pane is in the map selector stage, which contains all map list in the game.

/* You must implement a constructor in this class */

Method

Method	Description
- MapSelectorPane()	/* TODO: Complete the remaining code */ Constructor which initializes new MapSelectorPane. ● First, initialize the superclass by calling superclass' constructor. ● Set the preferred width and height to 400 and 600 pixels respectively. ● Set the alignment of the pane to **Pos.TOP_CENTER**. ● Set the spacing of the pane to 10 pixels. ● Set the background color of the pane to **Color.web("#EEF7FF").** See class **CellPane** for examples. ● Set the padding of the pane at top and bottom for 10 pixels. Other parameters for the Insets(top,right,bottom,left) are 0. ● For each map, add a **new MapButtonPane(map)** to the children of **this** MapSelectorPane.
+ MapSelectorPane getInstance()	Return the singleton instance of MapSelectorPane.

3.3.3 class MapSelectorStage extends Stage

This is the secondary stage for the map selector pane. This will show when the player click select map button.

3.4 package io

This package is responsible for reading map file in resource folder.

3.4.1 class MapParser

This class is the parser for reading the map files.

3.5 package logic

This package contains the logic of game system.

3.5.1 class GameSystem

This class is a game system that will control everything in the game.

3.5.2 class Map

This class represents the map of the game.

Field

Field	Description
- String name	The name of the map
- int width	The width of the map
- int height	The height of the map
- int maxCommandCount	The maximum commands allow for the map
- List<List<Integer>> tiles	The 2-dimensional map tiles list
- int spawnRow	The row that player spawns when starting the map
- int spawnCol	The column that player spawns when starting the map
- int difficulty	The difficulty level of the map

Method

Method	Description
+ Map(String name, int width, int height, int maxCommandCount, List<List<Integer>> tiles, int spawnRow, int spawnCol, int difficulty)	Constructor which initializes the map
Other getters and setters	

3.5.3 class Player

This class represents the player in the game.

3.5.4 enum Command

This enum represents the command type, including UP, DOWN, LEFT, and RIGHT.

3.5.5 enum GameState

This enum represents the state of the game, including STOPPING, PLAYING, and RUNNING.

3.5.6 enum Tile

This enum represents the type of tile, including DEER, WALL, GROUND, SHIKASENBEI, and EMPTY.

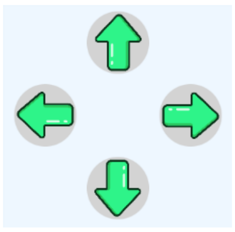
4. Scoring Criteria (12 points, will be scaled to 10 points) See demo.mp4 for reference.

CommandArrowPane

- (2 points) Each command is added successfully after clicking on each command arrow pane.

CommandControllerPane

- (2 points) The arrows to input commands are shown correctly. 0.5 points for each arrow.



GamePane

- (1 points) The size of the grid is correct after selecting map.
- (2 points) The tiles are shown correctly (without swapping row and column) in the game pane after selecting map.

MapSelectorPane

- (2 points) The style, background color, padding, alignment, and spacing of the pane is correct.
- (1 point) The buttons for each map are correctly displayed with correct order.

JAR File

- (2 points) The exported JAR file is runnable (all images and icon must be shown correctly).

  Export jar (**q5.jar**) file that **includes source code, resources (image and maps), and .class files** and put it in **root** folder of your project. **Your jar file must be runnable**. <u>YOU MUST EXPORT JAR FILE containing all mentioned contents. IF YOUR FILE DOES NOT CONTAIN EVEN 1 MENTIONED FILE, YOUR CODE WILL NOT BE MARKED</u>.