Project:
A JavaFX Implementation

1 Key Submission Details

Deadline: Dec 3th 2023, 5pm

Plagiarism: The submission must be yours and yours alone. If you are unsure what is or is not plagiarism, the following is a none exhaustive list of example activities you cannot do:

- Copy the completed files of another group and submit them as your own
- Share copies, images or print outs of your code with another group (by e-mail, FB messenger, WhatsApp etc.)
- Groups of students working on a single solution and then all submitting the same work or subset of the same work (regardless of whether variable, method, class names or ordering have been changed)
- Groups collaborating at too detailed a level. For example, consulting each other after each line / block / segment of code and/or sharing the results.

The game is a board-based game which has a start line and a finish line, and a course of obstacles between the two. The goal is easy: the player to get from the start to the end first is the winner. However, there are a few rules that make this a little more complicated! The rules which determine if a move is valid are:

1. A 9-sided dice is rolled to determine how many squares a player can move: 1, 2, ..., 9 squares.
2. Players always move forwards, but might have to interact with obstacles on the way. You should have controls to allow players to decide if they want to move, or stop.
3. If a player cannot make a valid move, they stop at the last valid square they can.
4. Obstructions include: other players, the edges / walls of the board, and obstacles in the game (fires, fences, pits, teleporting portals etc.).

To provide some context, consider the setup displayed in Figure 1. Here, 2 or more players, race to the finish. The board is arranged as a spiral, the start line is on the outside, the first player to get to the centre square wins. The brown lines indicate the walls that define the path from start to finish. While in Figure 1 there is only one path, you could consider extending the game to facilitate multiple paths.

There are, in this example, four types of obstruction:

- a bottomless pit (black background): causing the player to return to the start if they fall in (land on it)
- a fire pit (blue background): causing the player to miss a turn after they pass it
- a spike pit: must be jumped over, i.e. if it consumes 3 tiles, at least 4 is needed on the dice to pass the obstacle by the time the player reaches it
- a teleporter (white background): moves either the current player, or a player of their choice to a random location as soon as the player enters the tile (regardless of how much further they could move).

Note this board set up is only indicative, your board:

- could have more obstacles,
- could have perhaps more obstacle types,
- will likely be larger (have more squares), and
- perhaps have more than one route to the centre

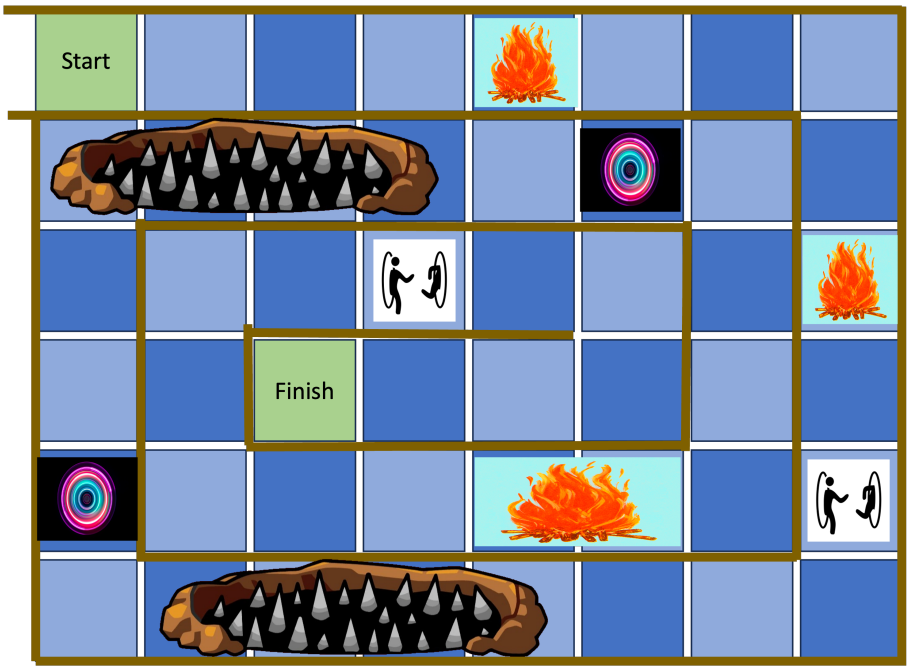

Figure 1: Example Game

Note the following:

1. you need 2 or more players, but 2 players cannot be on the same square
2. players must be able to make decisions when needed (you can decide how they do this)
3. you can have as large a board (i.e. the distance between start and end) as you like
4. the board should fit on the screen in at least one direction, i.e. either the full width or full height of the board should be visible, the other direction can scroll if needed.
5. you can have as many obstacles as you like, however, there are some minimum requirements

There is a lot of free design space for you, your game can look very different to the mockup shown in Figure 1: it is only a quick illustration. There is no legitimate reason for two projects to look the same, and as such two projects that are "too similar" may be scrutinised more closely for issues of plagiarism.

Requirements

In this assignment you are making a multi-player game, however, the following **MINIMUM** requirements should be observed:

1. All conditions must be observed
2. The game **MUST** have a 2D JavaFX user interface (which can be quite simple: fancy animations are **NOT NEEDED!**)
3. When a player wins, the game should end immediately and "announce" the winner
4. The players should be able to specify their name, this should be displayed in the interface
5. The interface should illustrate current value of the game dice
6. The game must be well unit tested
7. this **IS NOT** a maze game, and should not be constructed as a maze, the spiral shape should **ALWAYS** be retained and clearly visible.
8. You need to derive a scoring mechanism for the game, and have a persistent top 10 score board (e.g. in a flat file, serialised object, etc.);
9. all code should be placed within a well-structured Java package with accompanying JavaDoc that provides **detailed information** for all classes, methods, fields etc. Simply making a JavaDoc with no informative content will receive a score of zero.
10. you must have at least 3 different types of obstacle, one of these should have "special rules" i.e. modify the game play in some manner, e.g. a button that when pressed (stepped on) rearranges the board, a fire pit that makes the player miss their next turn, a teleportation portal that swaps the position of both players, etc.

All these requirements must be fulfilled to achieve a B grade or higher. To increase your mark, you can add adaptions to the game that increase its complexity and allow you to go beyond the minimum requirements (above). **However, this should only be undertaken once all minimum requirements are met**. Example adaptions:

1. user-specified board sizes and layout (e.g. of obstacles)
2. a difficulty setting (more or less obstacles)
3. randomly generated boards (with clear rules and parameters for the construction of the board)
4. player defined board layouts
5. < add your own ideas here >

Grading Rubric

Criteria	A+, A, A-	B+, B, B-	C+, C, C-	D+, D, D-	≤ FM
Game Implementation (50%)	Class design is aligned with OO principles and evidences high cohesion. All minimum requirements met. One or more advanced features implemented. Game is functional, and evidences complexity.	Class design is commensurate with fundamental OO principles. All minimum requirements met. Game is functional, and evidences some complexity.	Class design is acceptable, but there would be better ways to implement the solution. OO principles are attempted. Most minimum requirements met. Game is functional, but simple.	Class design is acceptable, but there would be much better ways to implement the solution. Most minimum requirements met. Game mostly functional, but potentially a few features are unreliable or barely functioning.	Class design is poor. Minimum requirements not met. Game perhaps only partially functioning.
Testing (25%)	A very thorough set of unit tests are implemented resulting in a well tested game.	A thorough set of unit tests are implemented, yet some aspects of code coverage are limited.	A good attempt at unit testing all core functionality. Test cases capture both expected behaviour and some fringe cases.	Evidence of unit testing for most core functionality, but is not extensive.	Testing is superficial or (mostly) not present.
JavaDoc (15%)	A thorough JavaDoc is present for all classes, and captures all essential details. For key classes / methods, meaningful examples of how to use them are also present.	A JavaDoc is present for all classes, and captures all essential details.	A JavaDoc is present for all classes, and captures most essential details.	A JavaDoc is present for all classes, and captures essential detail, but lacks depth.	Superficial or not present.
Video (10%)	A well-conceived video demonstrating all key functionality.	A well-conceived video demonstrating key functionality.	A well-conceived video demonstrating a functioning game.	A demonstration video is provided that shows a functioning game. However, the video is poorly conceived and/or lacks depth.	A demonstration video may be provided, but is poorly conceived or does not clearly illustrate a functioning game.