Homework 09 – Jungeon Jrawler

Authors: Kellen, Eric, Xavier
Topics: JavaFX

Problem Description

In this assignment, you will be implementing your own version of a Dungeon Crawler using ***J***avaFX (***J***ungeon ***J***rawler)!

For this homework, there will be a few basic requirements with certain files already provided, but the UI and design of the game is largely up to you! Feel free to explore some fun aspects of JavaFX. This homework is intentionally more open-ended, and we will grade most of it manually, so please try to have fun!

Solution Description

You will need to complete and turn in 1 class: `JungeonJrawler.java`. This class will be the base for your GUI and you will need to meet the requirements listed below. Feel free to create additional classes to help make your code more readable and modular, but the base class for your GUI must be `JungeonJrawler.java`.

JavaFX Details

- To compile a JavaFX program, run this command:
  - `javac --module-path javafx-sdk-11.0.2/lib/ --add-modules=javafx.controls FileName.java`
- To run a JavaFX program, run this command:
  - `java --module-path javafx-sdk-11.0.2/lib/ --add-modules=javafx.controls FileName`
- Modify the module path if your JavaFX download is not located in the same folder.

Backend.java (Provided File)

This class represents the backend implementation of your JungeonJrawler game and contains the game logic used in checking for collisions with wall and enemy, to obtain coordinates of the enemies chasing the player, and to check if the exit door has been reached. There are also some static methods to help simplify the setup of the game.

Note: This file should not be modified and is not required to be used. However, it should simplify a lot of the game logic if it is used.

Constructors:

- `Backend(List<Rectangle>, Rectangle, Rectangle, Rectangle, Rectangle)`
  - Takes in the list of walls, the goal rectangle, the player rectangle, and two enemy rectangles, in that order. This initializes the backend to enable the game logic.

Instance Methods:

- `wallCollision(double, double)`
  - Takes in x and y coordinates representing the new position of the player, and checking if getting into that position will cause a collision with a wall.
- `enemyCollision(double newX, double newY)`
  - Takes in x and y coordinates representing the new position of the player, and checking if getting into that position will cause a collision with an enemy.
- `enemy1ChasePlayer()`
  - Returns an array of length 2 representing [x, y] coordinates of the updated position for enemy1 when chasing the player.
- `Enemy2ChasePlayer()`
  - Returns an array of length 2 representing [x, y] coordinates of the updated position for enemy2 when chasing the player.
- `goalReached()`
  - Returns a boolean representing whether the player has reached the exit door.

Static Methods:

- `buildMaze(Color)`
  - Builds a maze with walls of the specified color and returns it as a list of rectangles.
  - Note that the dimensions of the scene displaying these walls should have a width of at least 850 and a height of at least 650. You may need to further increase the dimensions depending on the layout of other visual components.
- `generateRandomName()`
  - Generates a random name with a length of 2, 3, or 4.

JungeonJrawler.java

Write a JavaFX application class called `JungeonJrawler`. This class must meet the following minimum requirements:

- Create a window with the title "`JungeonJrawler`". The window should be sized appropriately, such that contents within the window are not cut off.
- The application will have three screens – the welcome screen, character naming screen, and game screen.
  - **Welcome Screen**
    - This is where the player will be greeted.
    - There should be a non-editable display of text that says "`JungeonJrawler`".
    - There should be an appropriate background image that relates to your Jungeon Jrawler. Name this file `welcomeImage`.
      - If you don't want to find your own image, we have provided one for you.
      - **IMPORTANT: Make sure to use relative paths for images in your application. Not using relative paths will result in the image not showing on our end when we grade your submission!**
    - There should be a button that says "`Play`", which when pressed, takes the user to the character naming screen.

- **Character Naming Screen**
  - This is where the player can enter their name.
  - There should be a non-editable display of text that says, "`Who am I?`"
  - There should be an appropriate background image that relates to your Jungeon Jrawler. Name this file `characterImage`.
    - If you don't want to find your own image, we have provided one for you.
    - **IMPORTANT: Make sure to use relative paths for images in your application. Not using relative paths will result in the image not showing on our end when we grade your submission!**
  - There should be a text input field that has "`Enter name here...`" as prompt text.
  - There should be a button that says "`Begin Jrawl`", which when pressed, stores the user's input in the text input field as the player's name and takes the user to the game screen.
    - If the user's input is blank, an alert that informs the user of the issue should be displayed.
- **Game Screen**
  - This is where JungeonJrawler will be played.
  - You should create an instance of a `Backend`.
  - You should Include an instructions button that, when clicked, opens a **new window** with a list of written instructions for how to play JungeonJrawler.
    - Opening the new window should not close the game screen.
    - So long as the Instructions window is open, the user should not be able to interact with the game screen.
      - **Hint:** A modal stage prevents events from being delivered to any other application window. See the Modality section in the Stage API.
  - **Maze**
    - Include a simple maze layout which the player will have to traverse to get to a door to leave and win the game.
      - The maze should have at least three bends and one main room.
        - **Hint:** Use the provided `buildMaze(Color)` method in Backend to build the maze.
      - The player should not be able to move through the walls.
  - **Player Creation**
    - The player should be a colored rectangle on the screen, initially located at the bottom right of the maze.
    - The player's name should be displayed above the rectangle representing the player.
    - The player's name should be the name taken from the text input field on the character naming screen.

- **Player Controls**
  - The player should be able to move around the screen using the following scheme:
    - Press '**W**' to move up
    - Press '**A**' to move left
    - Press '**S**' to move down
    - Press '**D**' to move right
- **Player Health Bar Creation**
  - There should be a health bar portraying the player's current health on the screen denoted as the number of lives it has.
  - The player should begin with five lives.
  - This health bar should consist of:
    - Shapes whose quantity represents the number of lives remaining for the player (e.g., three rectangles for three lives remaining).
    - A non-editable display of text that says, `"Health Bar"`, or something similar.
  - When the player loses a life, the count of these shapes should decrease by one.
  - If a player has no lives remaining, the game screen should close and `"You lost!"` should be printed to the console.
  - A player should lose a life when they collide with an enemy.
  - After colliding with an enemy, the player should not be able to lose another life for two seconds.
    - **Hint:** You can use an instance of the **`PauseTransition`** class combined with some state variable(s) to implement this functionality. Its constructor takes in a **`Duration`** object and you can use the static `seconds(double s)` method to define a duration of `s` seconds. Then, use the inherited `setOnFinished(EventHandler<ActionEvent>)` method to define what happens after happens after the specified duration has elapsed. Finally, call `play()` to start the countdown.
- **Exit (Goal/Door) Creation:**
  - There should be a rectangle at the top right of the maze acting as the door to escape.
  - When the player collides with this rectangle, the game screen should close and `"Congrats you won!"` should be printed to the console.
- **Enemy Creation**
  - There should be two enemies on screen located in the main room of the maze, both visually represented by colored rectangles (this color should be different than the player).
  - Each enemy should have its own randomly generated name displayed above it.
  - Each enemy's name should be randomly generated.
    - **Hint:** Use the static `generateRandomName()` method provided in Backend to generate a random name.

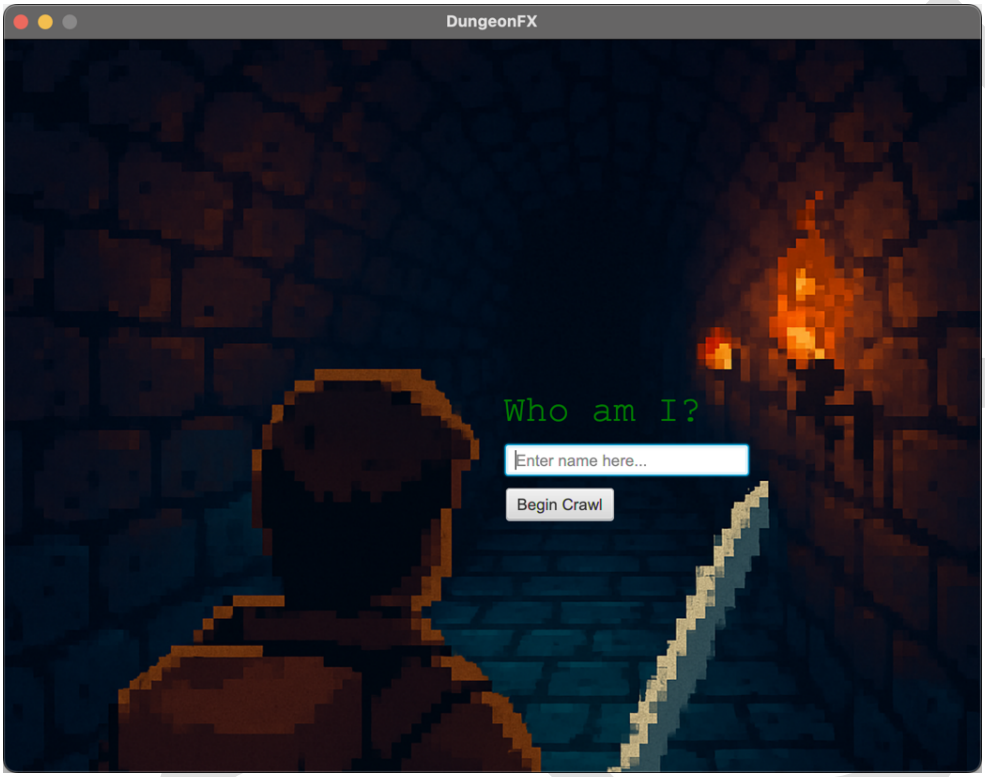
- **Game Loop**
  - To implement this gameplay screen properly, we can use an instance of the **`AnimationTimer`** class to rapidly "loop" the execution of some code.
    - This is an abstract class that has an abstract method named `handle`. You can write a regular class or an anonymous inner class (recommended) to override this method. The body of this method defines what will be rapidly looped.
    - Once you have the AnimationTimer object, you can call `start()` to start the loop or `stop()` to stop the loop.
    - You can use the handle method to update positions of the player and enemy rectangles, check for collisions, and check if the game has ended, among other things.
- When creating your implementation, use at least one anonymous inner class and one lambda expression in your implementation. (This is required for full points).

Screenshots

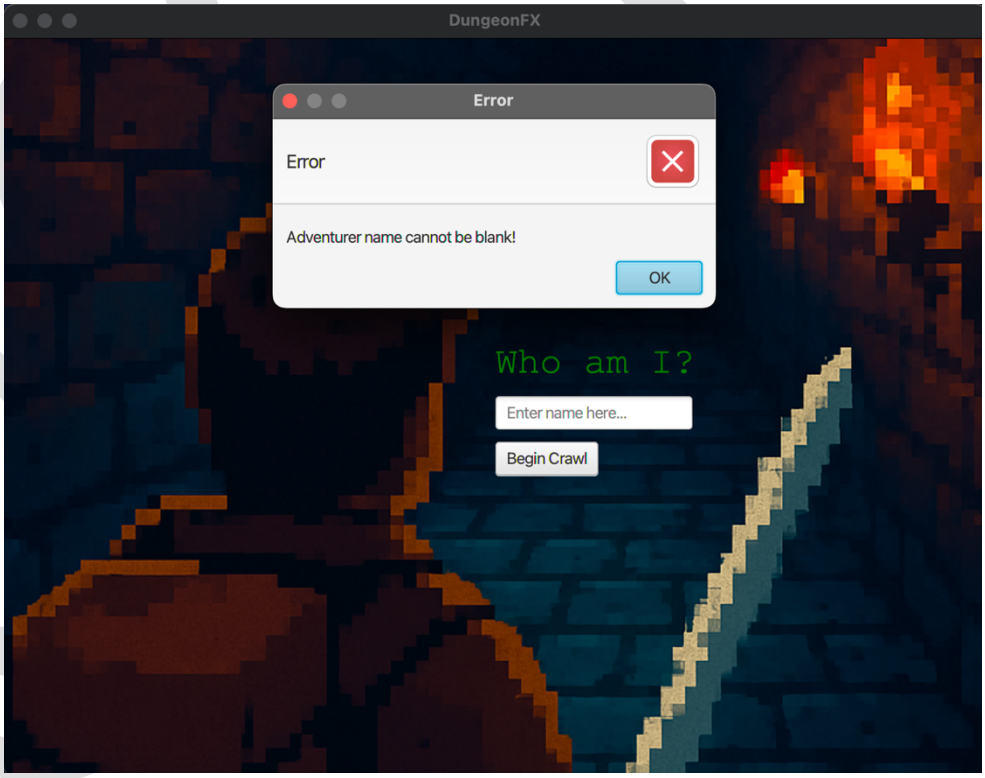
Below are sample images of a Jungeon Jrawler which correctly meets the requirements above. You may style your game just like this, or differently, but make sure to fulfill the requirements above.

Welcome Screen

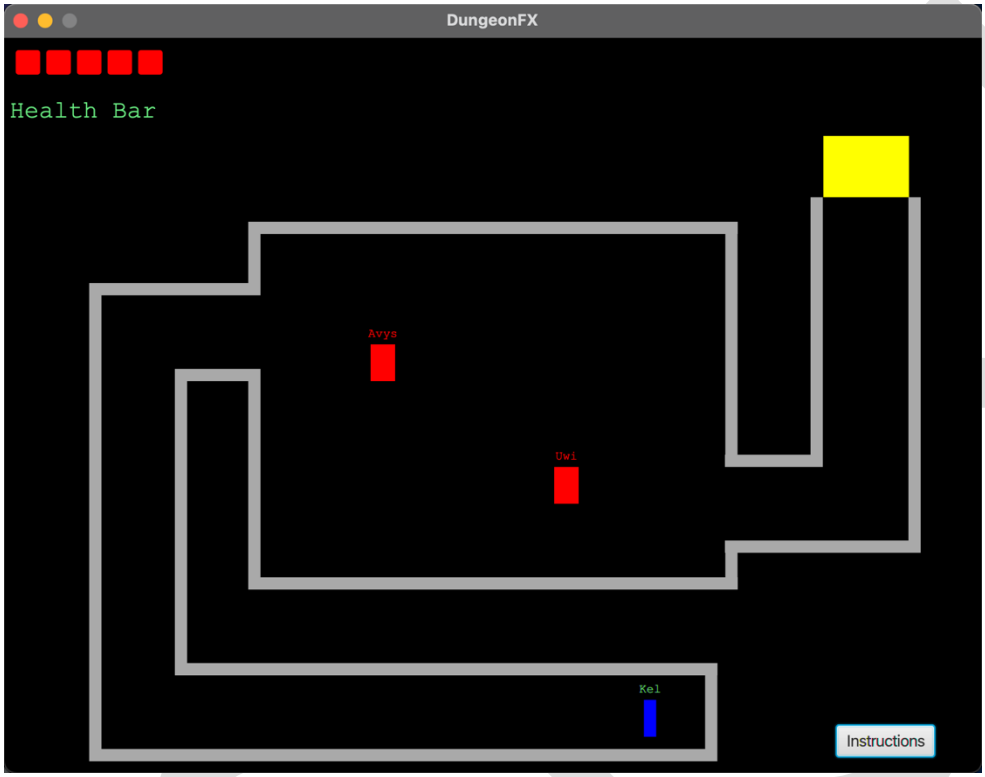
Character Naming Screen



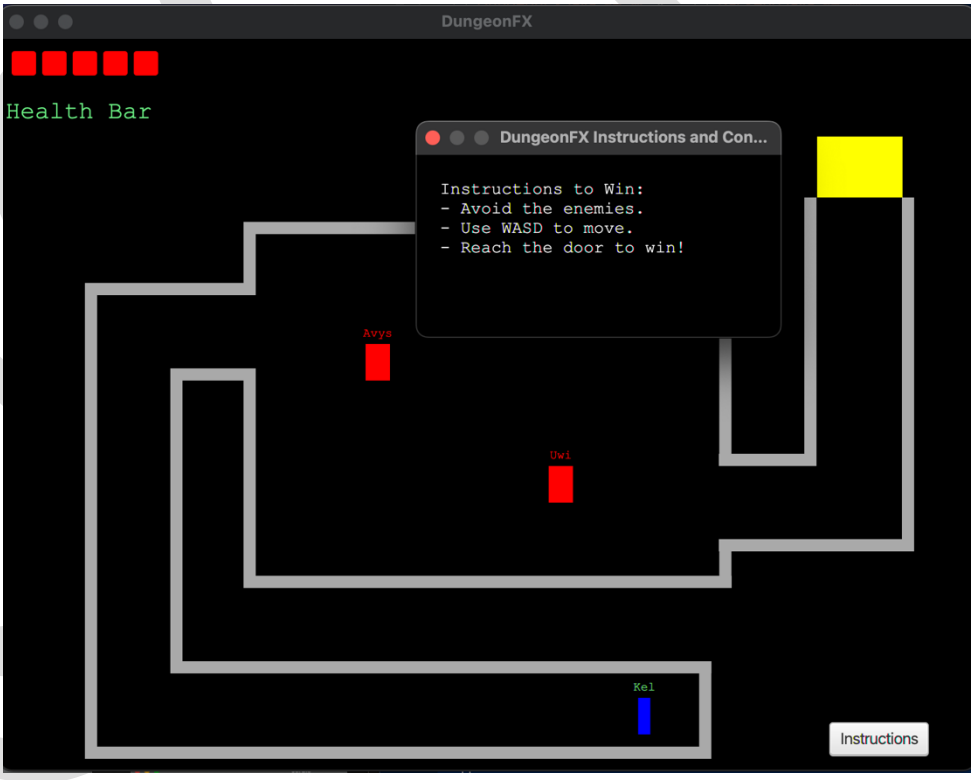
Alert: User Input is Empty



Game Screen



Instruction Window



JavaFX Tips and Tricks

- The Java API is your friend. Use it to your advantage.
- Make sure you are able to properly run your JavaFX programs before starting this assignment. Do **NOT** wait until the last minute to configure your JavaFX!
- Work incrementally. Get a basic UI up and running. Add buttons, cells for display, etc., piece by piece. Think of what layout manager(s) you want to use.

Extra Credit

As this homework is relatively open-ended, there will be up to **30 bonus points available** for expressing creativity through your implementation.

Note: If you decide to do extra credit, also submit an `extra_credit.txt` file detailing what extra credit you decided to implement (a few sentences or so). We will **NOT accept regrades or grade extra credit** if the `extra_credit.txt` file is not provided!

If your code requires a different command to compile and run (i.e., using different javafx modules), then put the command in the `extra_credit.txt`*.*

All these items are subjectively graded. Additions will be considered under the following categories. You may earn points under a certain category multiple times:

- 5 – Non-trivial enhancement to graphics of the program
- 5 to 15 – Non-trivial enhancement to the controls of the program
- 5 to 15 – Non-trivial new feature added

Some examples include, but are certainly not limited to:

- Prevent the enemies from being able to travel through the walls of the maze.
- Add a game over scene that takes the place of the `"You lose!"` and/or `"Congrats you won!"`.
- Audio feedback for the player losing health, reaching the door, moving on within the maze.
- Background audio acting as a soundtrack for each scene.
- Adding appropriate images and/or graphics in place of the simple rectangles for the player and enemies.

Feel free to be creative; these are just a few ideas!

Checkstyle

You must run Checkstyle on your submission (to learn more about Checkstyle, check out cs1331-style-guide.pdf under the Checkstyle Resources module on Canvas). **The Checkstyle cap for this assignment is 50 points.** This means there is a maximum point deduction of 50. If you don't have Checkstyle yet, download it from Canvas → Modules → Checkstyle Resources → checkstyle-8.28.jar. Place it in the same folder as the files you want to run Checkstyle on. Run Checkstyle on your code like so:

```
$ java -jar checkstyle-8.28.jar YourFileName.java
Starting audit...
Audit done.
```

The message above means there were no Checkstyle errors. If you had any errors, they would show up above this message, and the number at the end would be the number of points we would take off (limited by the Checkstyle cap). In future assignments we will be increasing this cap, so get into the habit of fixing these style errors early!

Additionally, you must Javadoc your code.

Run the following to only check your Javadocs:

```
$ java -jar checkstyle-8.28.jar -j yourFileName.java
```

Run the following to check both Javadocs and Checkstyle:

```
$ java -jar checkstyle-8.28.jar -a yourFileName.java
```

For additional help with Checkstyle see the CS 1331 Style Guide.

Turn-In Procedure

Submission

To submit, upload the files listed below to the corresponding assignment on Gradescope:

- `JungeonJrawler.java`
- `welcomeImage.png`
- `characterImage.png`
- `extra_credit.txt` (only if you added extra credit features)
- **Any other files needed for your code to compile and run**

Make sure you see the message stating the assignment was submitted successfully. From this point, Gradescope will run a basic autograder on your submission as discussed in the next section. **Any autograder tests are provided as a courtesy to help "sanity check" your work and you may not see all the test cases used to grade your work.** You are responsible for thoroughly testing your submission on your own to ensure you have fulfilled the requirements of this assignment. If you have questions about the requirements given, reach out to a TA or Professor via the class forum for clarification.

You can submit as many times as you want before the deadline, so feel free to resubmit as you make substantial progress on the assignment. We will only grade your latest submission. **Be sure to submit every file each time you resubmit**.

Gradescope Autograder

If an autograder is enabled for this assignment, you may be able to see the results of a few basic test cases on your code. Typically, tests will correspond to a rubric item, and the score returned represents the performance of your code on those rubric items only. If you fail a test, you can look at the output to determine what went wrong and resubmit once you have fixed the issue. **We reserve the right to hide any or all test cases, so you should make sure to test your code thoroughly against the assignment's requirements.**

The Gradescope tests serve two main purposes:

- Prevent upload mistakes (e.g., non-compiling code)
- Provide basic formatting and usage validation

In other words, the test cases on Gradescope are by no means comprehensive. Be sure to thoroughly test your code by considering edge cases and writing your own test files. You also should avoid using Gradescope to compile, run, or Checkstyle your code; you can do that locally on your machine.

Other portions of your assignment can also be graded by a TA once the submission deadline has passed, so the output on Gradescope may not necessarily reflect your grade for the assignment.

Burden of Testing

You are responsible for thoroughly testing your submission against the written requirements to ensure you have fulfilled the requirements of this assignment.

Be **very careful** to note the way in which text output is formatted and spelled. Minor discrepancies could result in failed autograder cases.

If you have questions about the requirements given, reach out to a TA or Professor via the class forum for clarification.

Allowed Imports

The only modules you should use are:

- `javafx.controls`
- `javafx.media`

You may optionally import any other packages that starts with `java` or `javafx` that do not belong to the `AWT` or `Swing` APIs.

- **IMPORTANT Note: You are NOT allowed to use FXML or any of the associated classes or packages.**
- Note that JavaFX is different than AWT or Swing. If you are trying to import something from `javax.swing` or `java.awt`, you are probably importing the wrong class.

Feature Restrictions

There are a few features and methods in Java that overly simplify the concepts we are trying to teach or break our autograder. For that reason, do not use any of the following in your final submission:

- `var` (the reserved keyword)
- `System.exit`
- `System.arraycopy`

Collaboration

Only discussion of the assignment at a conceptual high level is allowed. You can discuss course concepts and assignments broadly; that is, at a conceptual level to increase your understanding. If you find yourself dropping to a level where specific Java code is being discussed, that is going too far. Those discussions should be reserved for the instructor and TAs. To be clear, you should never exchange code related to an assignment with anyone other than the instructor and TAs.

The only code you may share are test cases written in a Driver class. If you choose to share your Driver class, they should be posted to the assignment discussion thread on the course discussion forum. We encourage you to write test cases and share them with your classmates, but we will not verify their correctness (i.e., use them at your own risk).

You **MAY NOT** use code generation tools to complete this assignment. This includes generative AI tools such as ChatGPT.

Important Notes (Don't Skip)

- Non-compiling files will receive a 0 for all associated rubric items.
- Do not submit `.class` files.
- Test your code in addition to the basic checks on Gradescope.
- Submit every file each time you resubmit.
- Read the "Allowed Imports" and "Restricted Features" to avoid losing points.
- **Check on Ed Discussion for a note containing all official clarifications and sample outputs.**

It is expected that everyone will follow the Student-Faculty Expectations document and the Student Code of Conduct. The professor expects a **positive, respectful, and engaged academic environment** inside the classroom, outside the classroom, in all electronic communications, on all file submissions, and on any document submitted throughout the duration of the course. **No inappropriate language is to be used, and any assignment deemed by the professor to contain inappropriate offensive language or threats will get a zero.** You are to use professionalism in your work. Violations of this conduct policy will be turned over to the Office of Student Integrity for misconduct.