# Super Ghost Project

## General Overview

Super Ghost General Overview

## Programming Overview

Super Ghost Programmatic Overview Part 1
Super Ghost Programmatic Overview Part 2
Super Ghost Programmatic Overview Part 3
How to Start Ghost With Jdk 11

## Rule changes:

The premise of the game remains the same, there are two players adding words to a growing word fragment. Each player will take turns attempting to add letters to a growing word fragment. The letter you select should attempt to force your opponent to spell a word or create a word fragment that has no possibility of creating a word. Unlike the prior ghost competition, when it is your turn you are allowed to add a letter to either the front or back of the word fragment.

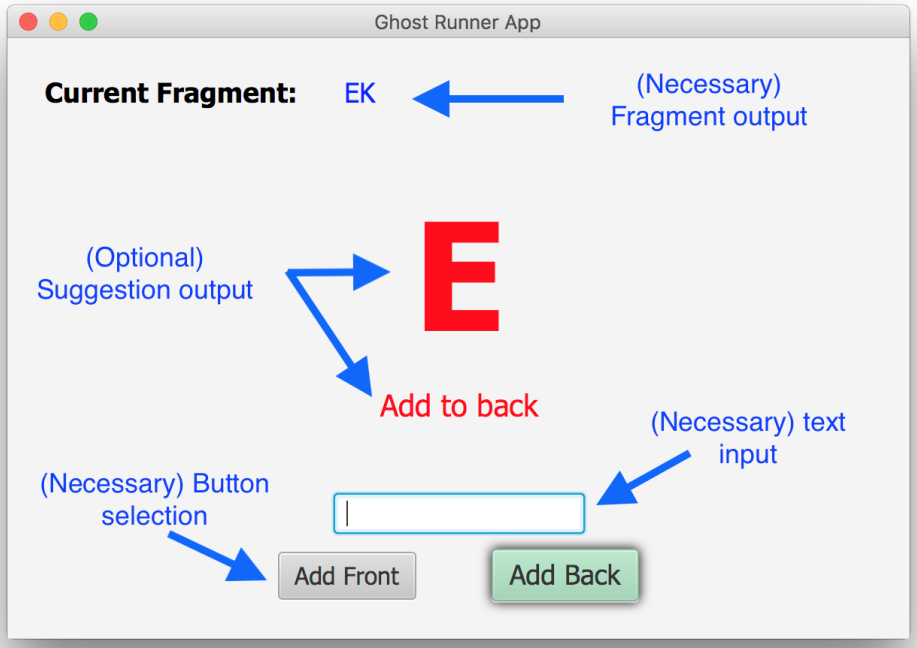
### Winning Criteria:

If your opponent spells a word that is at least 6 characters long
if your opponent creates a word fragment that has no possibility of creating a word
if your opponent takes longer than 60 seconds to select a word.
Tip: If you are the first player you can only spell words that are odd length thus you can only lose when you spell words that are odd but you can win when your opponent spells words that are even length. Accordingly, if you are the first player you should only care about spelling even length words. You can tell if you are the first player by the length of the word fragment when it is your turn...

## Programmatic Requirements:

The minimal requirement for this project is that you build a program that can save and load the share game data file with a GUI that enables you to control your team's actions. The GUI must contain a text field that displays the most current word fragment for each given round, it should update at the beginning of each round. It must also contain a text field where you can type in your next letter as well as some type of interface where you select the location the next letter

should be placed in the fragment.  Please use the accompanying photo as an example. Given that a portion of your grade is based on how well you perform in the competition it is highly recommended that you implement some type of assistance letter selection algorithm that aids you in your letter selection.




# Programmatic Changes:

Not much has changed the SGhostApp.jar will be used to start you and your opponent's program. SGhostApp.jar will act as the game arbiter that will stop the game if one of the winning criteria is met as well as notify either team of their turn to play by changing the game state in the shared game file. Below you will find documentation on the programmatic changes.

## TurnData:

This object holds information about the letter and location you intend to play. You will create this object in your button listener using the user inputted information from the text field as your letter and a particular button as the location. You create this class using the Turn Data's static function named "create". This function accepts three parameters, they are as follows: your application's team name, a character denoting the letter you intend to play and a boolean value signifying if you would like to add the letter to the front of the word or not (True = Front and False = Back).

## Letter Selection:

Since we now have the ability to add letters to the front and back of the word fragment your old logic of searching for words in your dictionary that start with the fragment will only partially get the job done. You must now look for words that contain the fragment.

## File I/O

We will not be writing to a text file like in the prior competition we will be writing to a binary file. The binary file will contain the class called ShareGameData.java. This class holds a list of GameActions and GameState. Use your IOManager to read and write to this file.

### GameActions

This class holds each player's action for each given turn. The action holds the letter that was played, the location where the letter was added, and the team the made the action. The game actions are stored in sequential order in the action record list of the ShareGameData class. Every time you play a new letter you will create a new GameAction and add it to the ShareGameData's action record, then you will save the file.

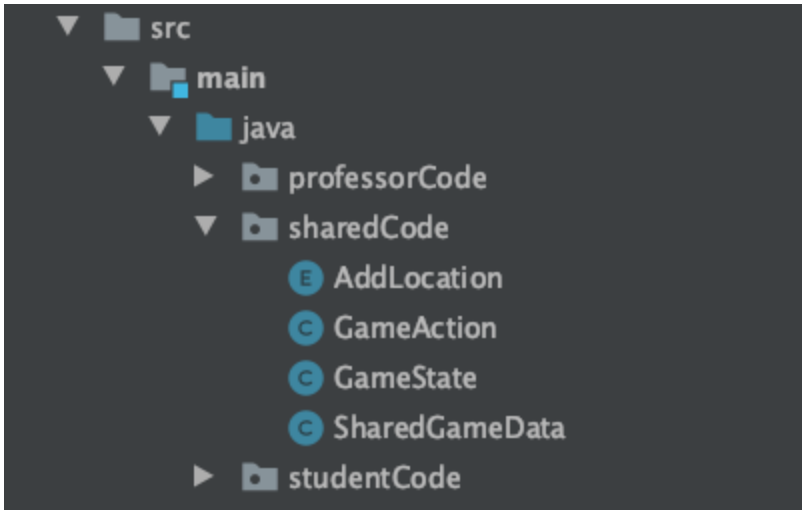
### GameState

This class holds information about the current game's state. This class is used to figure out if the game is over and who's turn it is. You should never change the values in this class. You should only be reading these values. This class will be changed by the GhostApp (The game arbiter)

### Project Organization (MANDATORY)

Any class that implements serializable must be put in a package called "sharedCode". Failure to do this will lead to a classNotFoundException being thrown by your IOManager when you attempt to read the sharedGameData file. The classes that must be put in the package named "sharedCode" package include: AddLocation.java, GameAction.java, GameState.java, and

SharedGameData.java. Below is an example of what my project structure looks like.




WARNING: You should not change any of the files that implement serializable!
What do you have to do?

1. Initialize your File Manager, Dictionary, IO Manager and Turn Parser in the Ghost Skeleton Class.
2. Implement the GameManager's constructor, onTurn and updateGUI method in the GameManager class. (See details below).
3. Create a jar file.
4. Submit.

# Programmatic Details:

You will be required to implement three methods in the GameManager.java file, the GameManager's constructor, the onTurn method and the updateGUI method (See documentation below). There are no restrictions on what dependencies you may use nor how many other classes you would like to add to this project.

- `GameManager(Stage primaryStage, IOManager ioManager, String teamName, int minWordLength, AbstractDictionary dictionary)` this is the constructor of the class. This method will be called once when your ghost player application is created for the first time. The method receives five parameters that you should use in your letter selection logic. These parameters should also be used to set up your GUI. Keep in mind this method is called once per game at the start of the game. It is suggested that you assign the minimum word length, your team name and the dictionary to local variables, so you may be able to access them during the game.
    - `primaryStage` - The javaFX main window, add your GUI widgets to this stage.
    - `ioManager` - The IO Manager you initialized in the Ghost Skeleton class. You will not need to access this during the game.
    - `teamName` - The file name of your jar file.

    - `minWordLength` - A value denoting the minimum length of the fragment that must be considered as a word. Furthermore, any word with fewer characters than the value is not considered a word and you will not lose if you spell a word that is this size or smaller. This will not change during the game.
    - `dictionary` - The dictionary class you initialized in the Ghost Skeleton class. Feel free to change the type to be your specific dictionary class so you can access the unique methods of your class.
- `onTurn(String fragment)` This method will be invoked when it is your turn to add a letter to a fragment. It will be invoked once per player turn. You will be given the current word fragment of the game as a parameter. This method will return the next letter to be added to the word fragment and its location. You are expected to perform logic using the word fragment, the minimum word length, and all the English words to select a letter and its location. When this function has completed you should save its selection in a local variable so you can access it in the Update GUI function. This function is NOT executed on the GUI thread, do not make any changes to your GUI in this function.
    - `fragment` - The current ordered collection of letters that have been played in the game.
- `updateGUI(String fragment)` This method is called when its time to update the GUI. It is called immediately after your onTurn function has successfully completed. This is where you should update your GUI with the necessary graphical changes as outlined in the GUI requirements of this project. This function is executed on the GUI thread, it is safe to make changes to your GUI in this function. Because this function is run on the GUI thread, any long-running tasks in this function will make your application GUI freeze.
    - `fragment` - The current ordered collection of letters that have been played in the game.
- `submitTurn(TurnData)` This method is already implemented for you. Put this function in your button listeners when you want to play a letter. It accepts a TurnData object that it will use when updating the shared game data.
    - `turnData` - A turn data object with the letter and location you would like to play. Create this object using the user-specified information in the text field and the button selected.

# Additional Material:

## SGhostApp.jar

Use this application to test your program. It allows you to play against your program and provides you with a GUI that will give you a suggestion for a letter and location to play (The suggestion is not randomized so you can trust these suggestions and use it to test the quality of your program). SGhostApp.jar will be used to in the actual competition if 2 programs are specified when the program is started the two programs will play against each other.

## To start SGhostApp.jar

Use the following command in terminal or command prompt: `java -jar [File Path to SGhostApp.jar] [File Path to a Ghost App] [Optional -File Path to a Ghost App (Defaults to user Testing Gui mode if not specified)]`

If you are working with java 9 or higher use the following command instead:

### Mac

```
java -jar --module-path $PATH_TO_FX --add-modules
javafx.controls,javafx.fxml [File Path to SGhostApp.jar] [File
Path to a Ghost App] [Optional -File Path to a Ghost App
(Defaults to user Testing Gui mode if not specified)]
```

### Windows

```
java -jar --module-path %PATH_TO_FX% --add-modules
javafx.controls,javafx.fxml [File Path to SGhostApp.jar] [File
Path to a Ghost App] [Optional -File Path to a Ghost App
(Defaults to user Testing Gui mode if not specified)]
```

The file paths outlined in the examples below are example file paths from the professor's computer you will most likely have to change the file paths to run the application on your computer.

### Java 8 and lower:

**Attention: If you are working on a PC replace "/" with "\"**

Example for user testing with a minimum length of 6 characters:

```
java -jar /path/to/SGhostApp.jar /path/to/player.jar
```

Example for a head-to-head competition:

```
java -jar /path/to/Desktop/SGhostApp.jar /path/to/player_1.jar
/path/to/player_2.jar
```

### Java 9 and up:

**Attention: you must have set up your JavaFX before attempting these commands see this video or slides for more information.**

Example for user testing with a minimum length of 6 characters:

Mac

```
java -jar --module-path $PATH_TO_FX --add-modules
javafx.controls,javafx.fxml /path/to/SGhostApp.jar
/path/to/player.jar
```

Windows

```
java -jar --module-path %PATH_TO_FX% --add-modules
javafx.controls,javafx.fxml \path\to\SGhostApp.jar
\path\to\player.jar
```

Example for a head to head competition:

*Mac*

```
java -jar --module-path $PATH_TO_FX --add-modules
javafx.controls,javafx.fxml /path/to/SGhostApp.jar
/path/to/player_1.jar /path/to/player_2.jar
```

*Windows*

```
java -jar --module-path %PATH_TO_FX% --add-modules
javafx.controls,javafx.fxml \path\to\SGhostApp.jar
\path\to/player_1.jar \path\to/player_2.jar
```

Example to display full documentation:

```
java -jar /path/to/SGhostApp.jar -h
```

# Grading

Since this competition will take place on the last day of class, there will be no possibility of any extensions. Please plan ahead and test your code thoroughly.
40% is based on if you pass in an assignment that opens and runs. Late submissions will not be accepted.
25% is based on if your application meets the GUI requirements outlined above.
25% is based on your application's programmatic performance. This means your application will be graded on how stable your code is and if it effectively plays the game. Simply submitting an application that selects random letters will not count, it needs to make a logical decision when selecting a letter.
10% is based on the results of your application. In other words how well your application performs verse the opponent applications.

Generally, if you pass something in and it works you shouldn't get lower than a B-

# Submitting

Submit your jar file and any additional files your application needs (such as your dictionary file) to this Moodle assignment page. You may submit just a jar file or a zip file containing your jar file and any additional necessary files. This assignment will not be accepted on Grader Than. Do NOT submit your source code, I will not read it. If you are using the same dictionary file (ARBITOR_DICTIOANRY.txt) as the Arbiter, you do not need to submit the dictionary file.

# Ghost Project

This is an empty Ghost project that you will import into your IDE to begin development of your final project. For more information on how to import this project into your IDEA please review the Technology and Resources Section in Module 0.

Link to empty Super Ghost Project