Week 5 Discussion

Wednesday, 10/29/19

Reminders

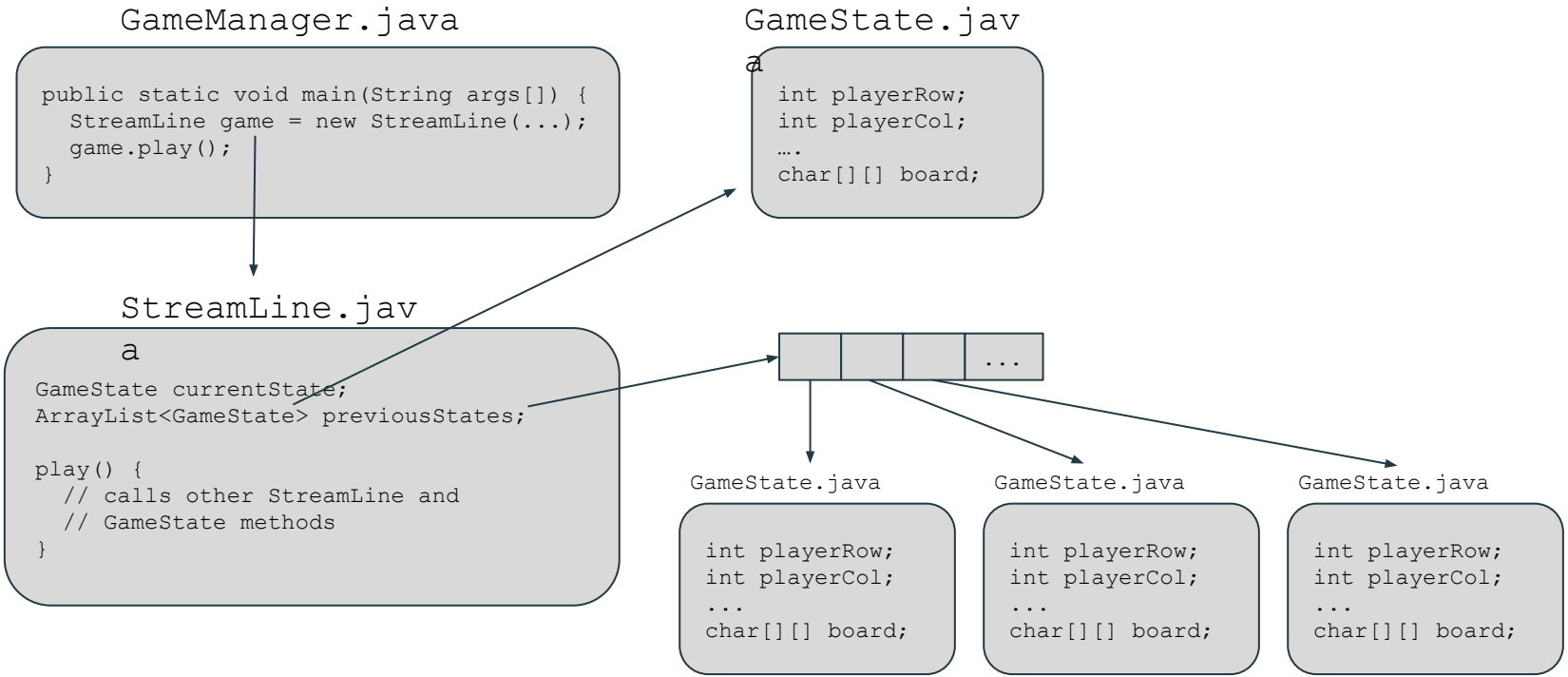
Quiz 2 on Monday, November 4

PSA3 Final Submission due Tuesday, November 5 11:59pm

Today’s agenda

- Quick high level overview of the files, specifically Streamline.java
- What you have to write
- There is no extra credit for this assignment.

How it all fits together



Streamline.java and its components

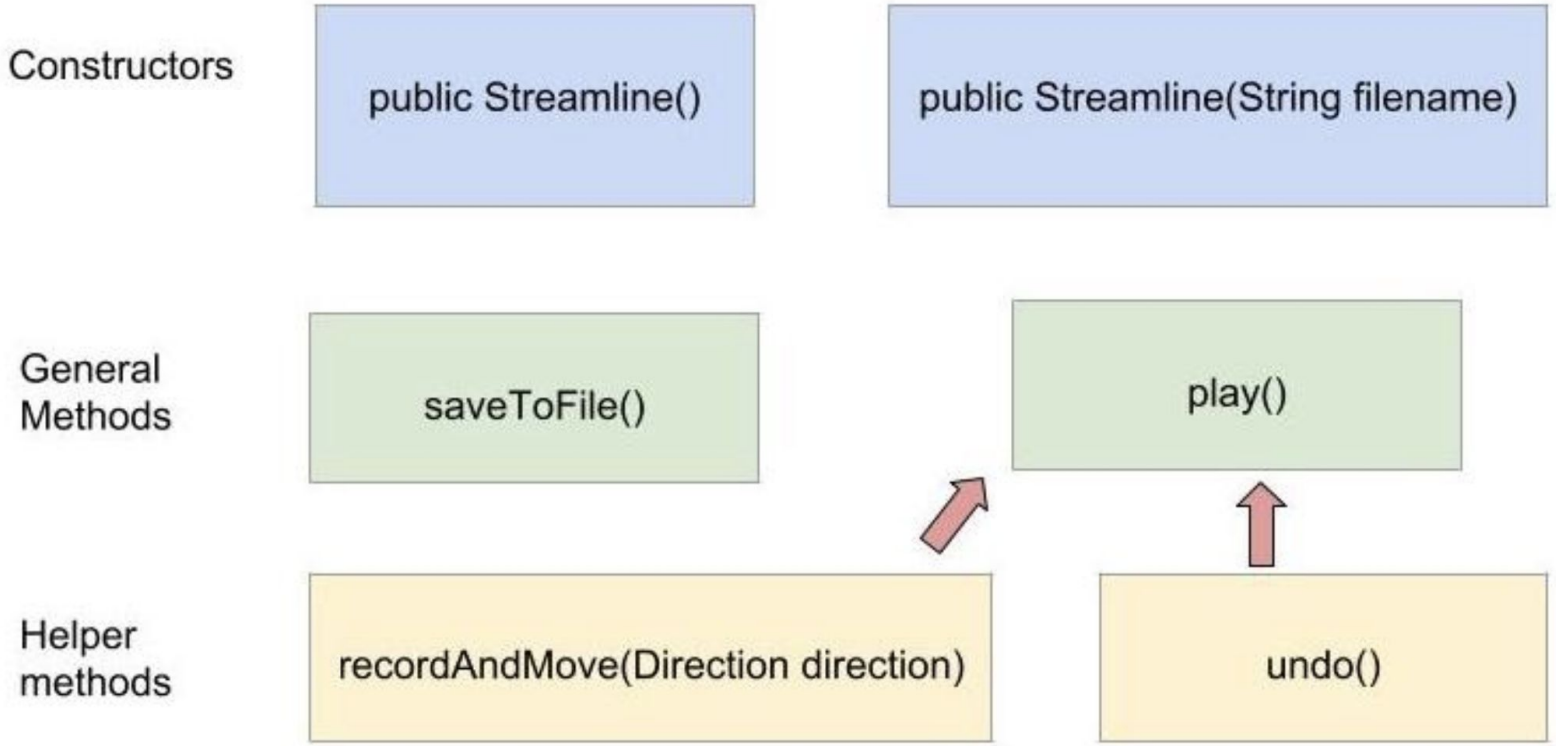
`GameState currentState;`

- Tracks the current state of the board

`List<GameState> previousStates;`

- Tracks all the previous states of the board
- This is what enables you to undo (will discuss later)

Streamline.java and its components



Constructors

`public Streamline()`

- Initialize the two instance variables
  - `GameState currentState;`
  - `List <GameState> previousStates;`
- Add 3 random obstacles and 3 random zappers to the board

Constructors

`public Streamline(String filename)`

- Initialize `currentState` based on information provided by filename (using `loadFromFile()`)

This is given to you in the write-up :D

`void play()`

- Indefinitely reads user input from the console until
  - Player passes the level OR
  - Player quits the game
- Things you should know how to do:
  - Read user input from the console (`Scanner`)
  - Switch (case) statements to do actions depending on what user inputted

Scanner: file I/O and console I/O

To read from the console:

- `Scanner inputReader = new Scanner(System.in);`

To read from a file:

- `Scanner fileReader = new Scanner(new File("someFile.txt"));`

Methods: `next()`, `nextInt()`, `nextLine()`, etc...

- Scanner documentation

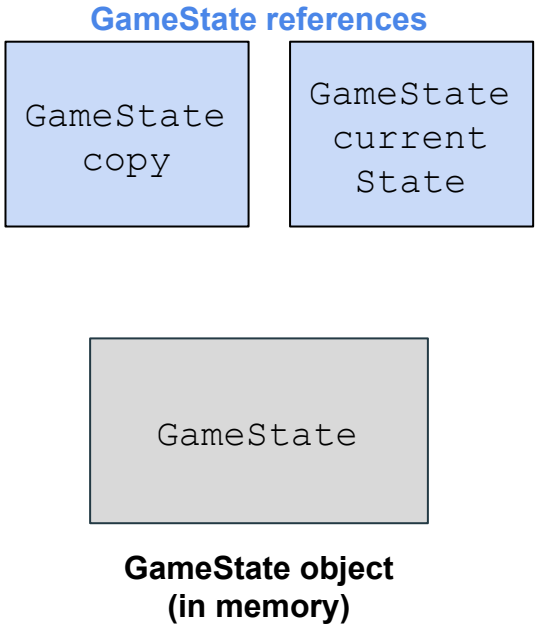
switch (case) statements

```
switch(value) {
  case value1:
    // statements (1)
    break;

  case value2:
    // statements (2)
    break;

  default:
    // statements (3)
}
```

```
if (value == value1) {
    // statements (1)
} else if (value == value2) {
    // statements (2)
} else {
    // statements (3)
}
```

`void recordAndMove(Direction direction)`

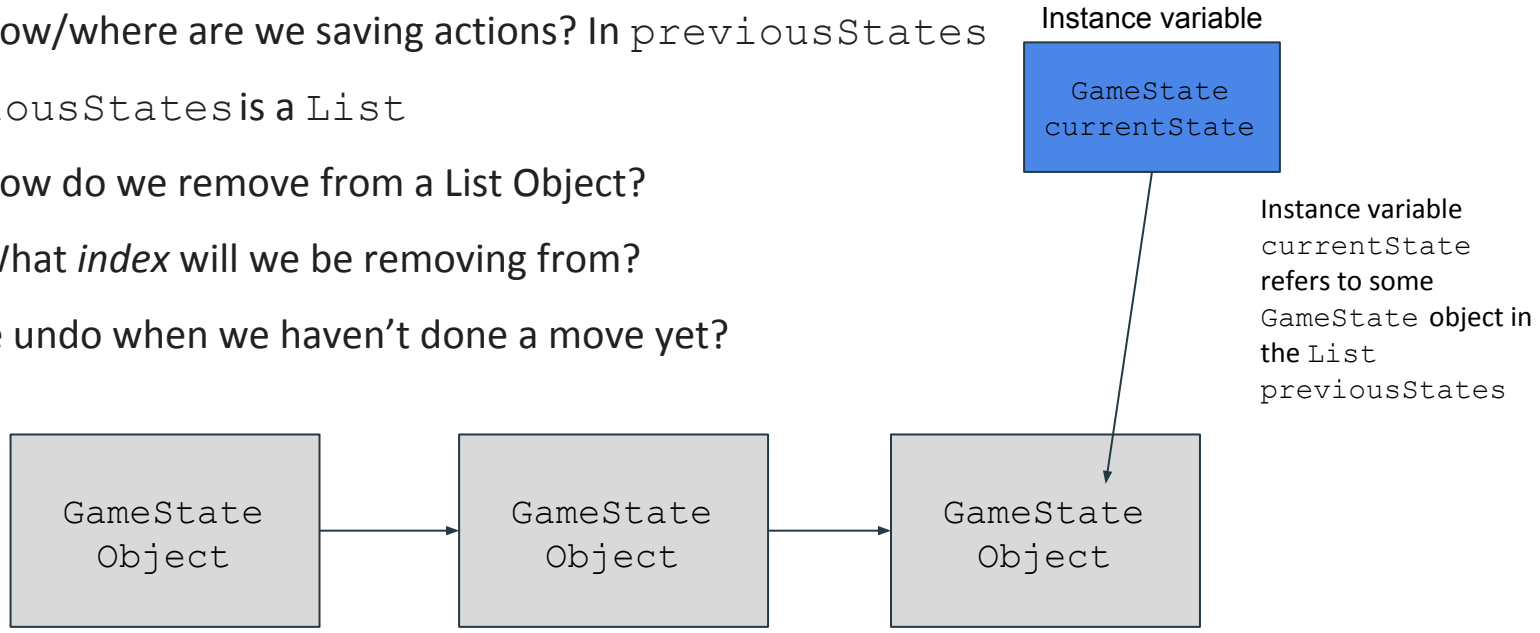
- Method should create a **copy** of current game state for manipulation
  - **Deep copy** vs. **shallow copy**
  - Did we make a constructor that lets us do this?
- Move the `GameState` based direction.
- What if direction is `null`?
- Check that the game state *actually* changed
  - How could you check equality?
- Save old `GameState` values to `previousStates`
  - `previousStates` is a `List`!
  - `Lists` that hold what kinds of `Objects`?
  - Where are we adding to in the lists?
- Set our `currentState` to the new `GameState`



Two references pointing to the *same object* in memory

void undo()

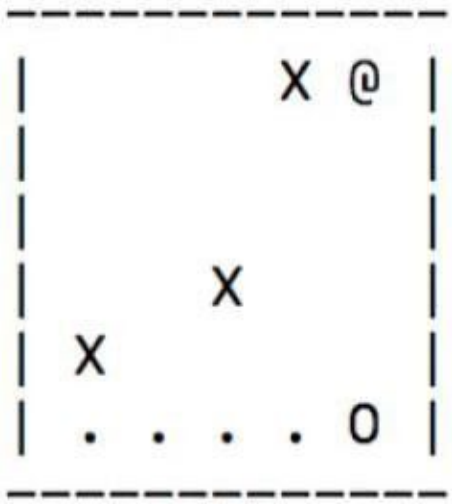
- Undo should undo the last taken action
  - How/where are we saving actions? In `previousStates`
- `previousStates` is a `List`
  - How do we remove from a List Object?
  - What *index* will we be removing from?
- Can we undo when we haven't done a move yet?



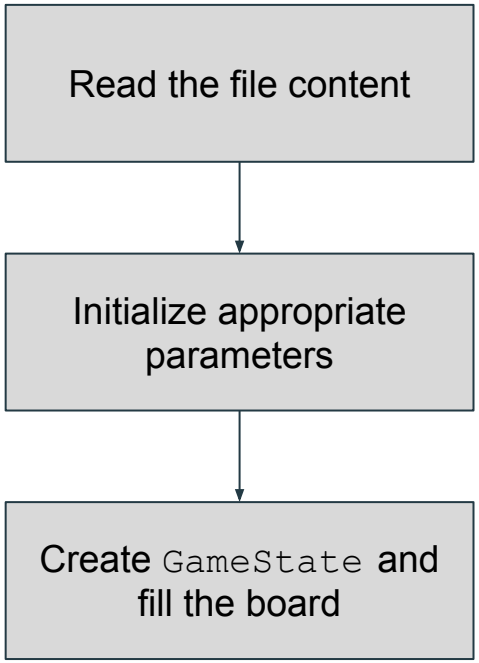
void loadFromFile(String filename) throws IOException

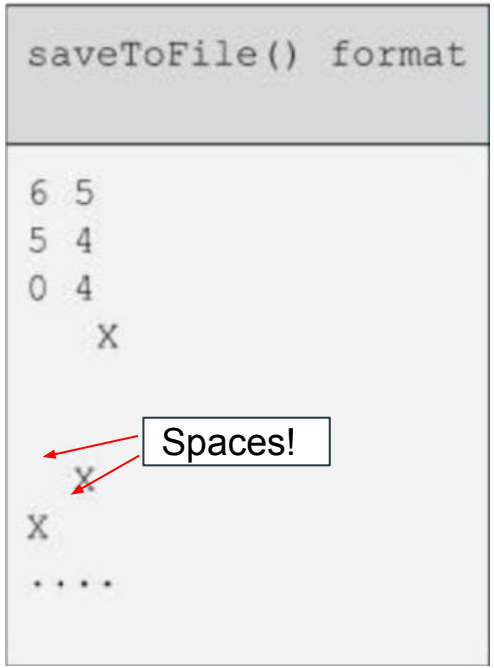
saveToFile() format	Explanation

6 5
5 4
0 4
  X
 	
  X
X
....
	<board length vertical, length horizontal>
	<player position vertical, horizontal>
	<goal position vertical, horizontal>
	<print rows and columns of the game state>



`void loadFromFile(String filename) throws IOException`





We have to read spaces instead of ignoring them.

Hint: Using `Scanner` class `next()` may not be a good idea.

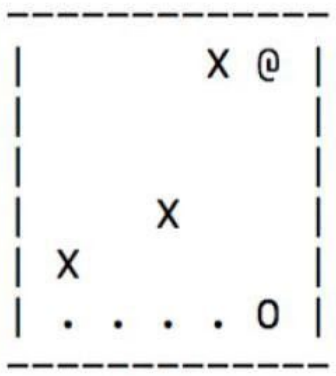
Another hint: What if we just read the whole line as a String and start from there?

void saveToFile()

- Write the game state to a file in an **EXACT** format as the diagram in `loadFromFile()`.
- Make sure the output matches exactly.
- Once finished, print a message to indicate that the file was saved successfully.
- Use `PrintWriter` to write to files.

saveToFile() format	Explanation

6 5
5 4
0 4
  X
 	
  X
X
....
	<board length vertical, length horizontal>
	<player position vertical, horizontal>
	<goal position vertical, horizontal>
	<print rows and columns of the game state>



PrintWriter: file I/O

- Prints formatted representations of objects to a text-output stream.
- Useful methods examples, refer to the documentation for more info.
  - Constructor: `PrintWriter(File file)`
  - Print a value: `print(int i)` prints an integer to the file
  - Print a line: `println(int i)` prints an integer AND terminates the line.

Quick Demo!

Style

Remember to have on **all files that you edit, INCLUDING testers**:

- File headers
- Class headers
- Method headers
- Inline comments
- Proper indentation
- Descriptive variables
- No magic numbers
    - Exception: You can have magic numbers in your tester files
- No lines over 80 characters
- Proper Javadoc convention

this vs no this

```
int x;

char y;

public Try() {

  y = 'a';

  x = 1;

}
```

```
int x;

char y;

public Try(int x, char y) {

  this.x = x;

  this.y = y;

}
```

```
int x; //instance

char y;

public Try(int x,char y) {

  x = x;

  y = y;

}
```

Be sure to differentiate between the local variables and instance variables.
Same goes for methods.