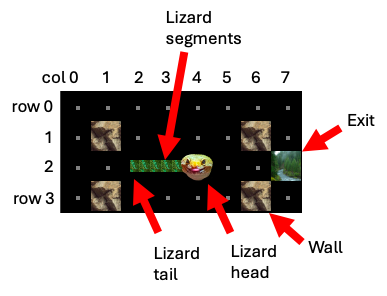
**LIZARD GAME**

There are three classes for you to implement: **LizardGame, Lizard, and GameFileUtil**. As always, your primary responsibility is to implement these classes according to the specification and test them carefully. The three classes can be used, along with some other components, to create an implementation of a game we call Lizards, which is a mix between the classic snake game1 and a sliding blocks puzzle game.

The game is played on a 2-dimensional grid of “cells”. Each cell may contain a wall, an exit, or the body segment of a lizard. Cells are located by column and row.

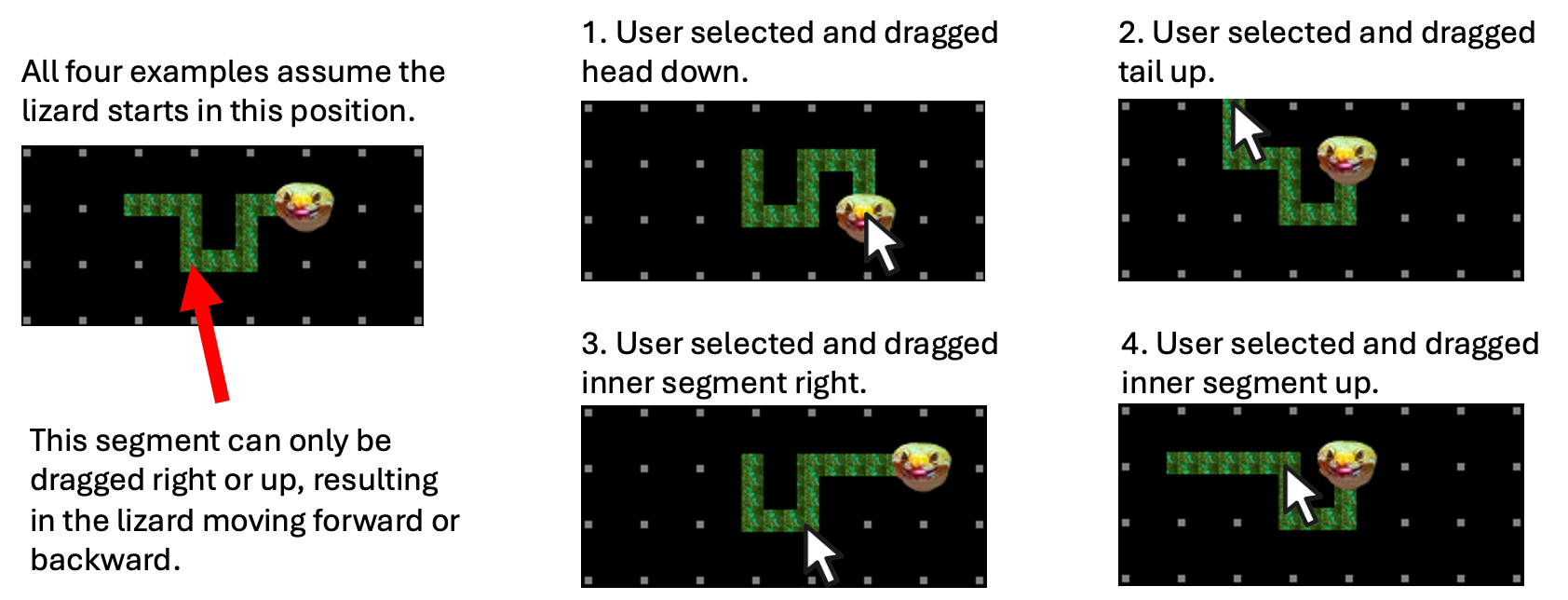


### Figure 1

The user presses down the left mouse button and drags lizards to move them around with the goal of getting all the lizards to the exit. A lizard’s body is multi-segmented and the user can press and drag any segment. The lizard moves in a snake-like fashion, that is, each segment must follow in the path of the segments in front or behind of them when the lizard moves forward or backward respectively. The only exception is when the user clicks and drags the head or tail

segments, which can move in any direction. There are only four possible directions of movement: up, down, right, left (never diagonally).

To illustrate the movement, Figure 2 shows four different scenarios: 1) the user selects the head segment and drags it in some forward direction 2) the user selects the tail segment and drags it in some backward direction 3) the user selects one of the inner segments and drags it in a forward direction 4) the user selects an inner segment and drags it in a backward direction. It is also possible, but not shown, that the user pushes the head segment in a backward direction or the tail segment in a forward direction.



### Figure 2

The lizard may not move off the grid and may not move onto a wall. When it moves onto an exit, the lizard is removed from the grid. When all of the lizards have been removed, the player wins.

The three classes you implement will provide the "backend" or core logic for the game. In the interest of having some fun with it, we will provide you with code for a GUI (graphical user interface), based on the Java Swing libraries.

The sample code includes a documented skeleton for the classes you are to create in the package. The additional code is in the packages **ui** and **api**. The **ui** package is the code for the GUI. The **api** package contains some classes for representing data in the game. There are some simple tests to start with located in the default package.

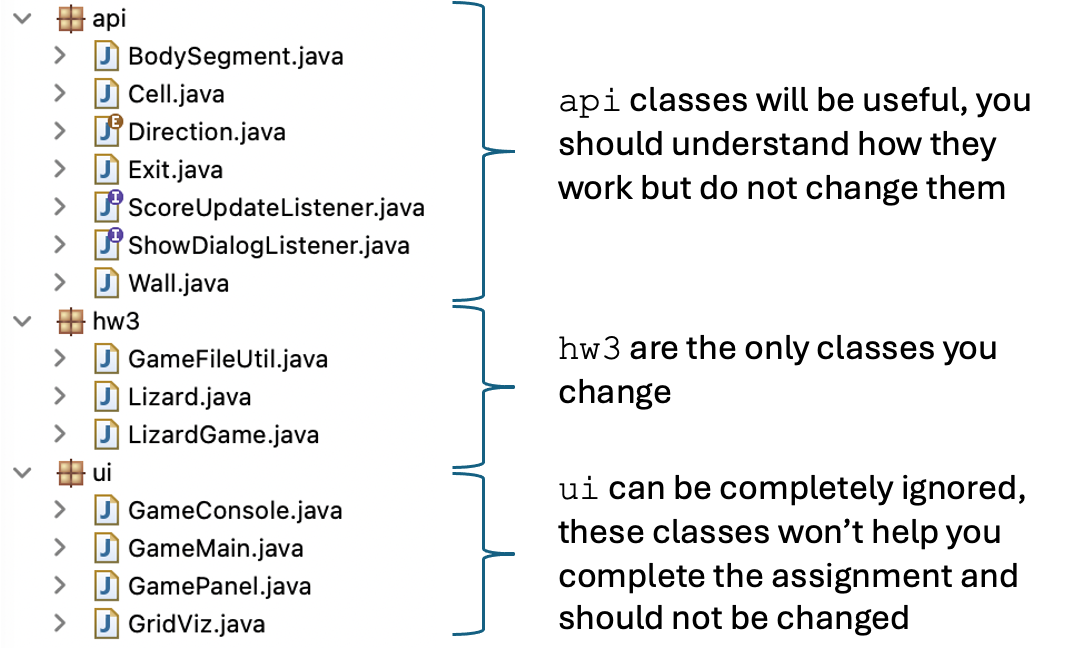


Figure 3

Do **not** modify the code in the **ui** or **api** packages.

# Overview of the GameFileUtil class

The **GameFileUtil** class is a simple utility class with a single static method to load a game file.

public static void load(String filePath, LizardGame game);

Game files are in plain text and have a particular format. There are three main sections to the file: the dimensions of the grid, a representation of the grid’s cells, and a description of the lizards. The table below gives an example of these three segments.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 8x8 | | | | | | | | | | | | | | | Number of columns by number of rows, can be any integers  larger than 0. |
| W WW .  WWW W.  . WWWWWW .  E. WWWWW W. W W.  W W . | | | | | | | | | | | | | | | A representation of the grid’s cells, showing the location of walls "W" and exits "E". Each row ends with a dot. |
| L L | 5,1  1,2 | 6,1  0,2 | 6,2  0,3 | 5,2  0,4 | 4,2  1,4 | 3,2  2,4 | 2,2  3,4 | 4,4 | 5,4 | 6,4 | 6,5 | 6,6 | 5,6 | 4,6 | Each line represents a lizard. The pairs of |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | numbers represent |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | the location of |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | segments, starting |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | from the tail (left |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | most pair) of the |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | lizard and ending |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | with the head (right |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | most). |

Coordinates are specified as (column, row) pairs. As shown in Figure 1, the columns are numbered from left to right starting at zero and the rows are numbered from top to bottom starting at zero. We use column before row consistently throughout this document and the API because it matches with the convention of specifying coordinates as (x, y), however, it conflicts with the convention of arrays being rows of columns. How you implement internal data structures such as arrays does not need to match with the API, all that matters is that you are consistent in your usage of the data structures.

Finally, note that load does not return anything. It is passed a LizardGame object and it is expected that the method calls mutator methods on the object to update the state of the game.

# Overview of the Lizard class

The **Lizard** class models a lizard as a collection of body segments (see the class **api.BodySegment**). The class contains several accessor and mutator method that can be used by other parts of the program, for example **LizardGame**. The segments are provided to the object with a call to **setSegments(ArrayList<BodySegment> segments)**, where the segments are expected to be ordered from tail to head. In other words, you can assume the first element of the list is always the tail.

# Overview of the LizardGame class

The **LizardGame** class models the game and its methods are called by classes in the **ui** package to create a complete game. For example, the UI calls **move()** every time the user presses and drags the mouse over the graphical representation of the gird.

Also, **LizardGame** is responsible for keeping track of the grid of cells. Each cell is represented as a **Cell** object. The grid is specified by (column, row), as shown in Figure 1 and described in **GameFileUtil** above, however you can store the cells any way you want as long as the public methods produce the correct results.

# Getting started:

A good starting point is the **Lizard** class. A lizard is modeled as a collection of body segments (see the class **api.BodySegment**). A lizard can be constructed for test like this.

**Lizard liz = new Lizard(); Cell cell0 = new Cell(1, 1); Cell cell1 = new Cell(2, 1); Cell cell2 = new Cell(2, 2);**

**ArrayList<BodySegment> segments = new ArrayList<BodySegment>(); segments.add(new BodySegment(liz, cell0));**

**segments.add(new BodySegment(liz, cell1)); segments.add(new BodySegment(liz, cell2));**

There are several methods to implement in **Lizard**. Right now, these methods may not seem to be that important. But when you implement **LizardGame**, you may find many uses for these methods, so take note of what they do. Here are some simple tests for the methods in **Lizard**.

**liz.setSegments(segments);**

**System.*out*.println("The lizard has " + liz.getSegments().size() + " segments, expected 3.");**

**BodySegment headSegment = liz.getHeadSegment(); Cell headCell = headSegment.getCell();**

**System.*out*.println("The head segment is at " + headCell + ", expected (2,2,Ground,Lizard).");**

**BodySegment tailSegment = liz.getTailSegment(); Cell tailCell = tailSegment.getCell();**

**System.*out*.println("The tail segment is at " + tailCell + ", expected (1,1,Ground,Lizard).");**

**BodySegment middleSegment = liz.getSegmentAt(cell1); Cell middleCell = middleSegment.getCell();**

**System.*out*.println("The middle segment is at " + middleCell + ", expected (2,1,Ground,Lizard).");**

**BodySegment aheadSegment = liz.getSegmentAhead(middleSegment); Cell aheadCell = aheadSegment.getCell();**

**System.*out*.println("The segment ahead of the middel is at " + aheadCell**

**+ ", expected (2,2,Ground,Lizard).");**

**BodySegment behindSegment = liz.getSegmentBehind(middleSegment); Cell behindCell = behindSegment.getCell();**

**System.*out*.println("The segment behind the middel is at " + behindCell**

**+ ", expected (1,1,Ground,Lizard).");**

**Direction aheadDir = liz.getDirectionToSegmentAhead(middleSegment); System.*out*.println("From the middle segment, ahead is " + aheadDir + ",**

**expected DOWN.");**

**Direction behindDir = liz.getDirectionToSegmentBehind(middleSegment);**

**System.*out*.println("From the middle segment, behind is " + behindDir + ", expected LEFT.");**

**Direction headDir = liz.getHeadDirection(); System.*out*.println("The head is pointing " + headDir + ", expected**

**DOWN.");**

**Direction tailDir = liz.getTailDirection(); System.*out*.println("The tail is pointing " + tailDir + ", expected**

**LEFT.");**

The class **GameFileUtil** is used to load saved game files. You are provided with a couple of sample files in the **examples** directory.

**// Example tests for GameFileUtil class**

**// (requires some implementation of LizardGame) LizardGame game = new LizardGame(0, 0); GameConsole gc = new GameConsole(); game.setListeners(gc, gc);**

**System.*out*.println(); GameFileUtil.*load*("examples/game1.txt", game);**

**System.*out*.println("Expected a message saying the number of lizards is now 1.");**

**System.*out*.println("DO NOT print this message in GameFileUtil, the ScoreListener needs to be called in LizardGame.");**

**System.*out*.println();**

**System.*out*.println("The grid with is " + game.getWidth() + ", expected**

**8.");**

**System.*out*.println("The grid height is " + game.getHeight() + ",**

**expected 4.");**

**System.*out*.println("The cell at (0,0) is empty (" + game.getCell(0, 0).isEmpty() + "), expected true.");**

**System.*out*.println("The cell at (1,1) has a wall (" + (game.getCell(1, 1).getWall() != null) + "), expected true.");**

**System.*out*.println("The cell at (7,2) has an exit (" + (game.getCell(7, 2).getExit() != null) + "), expected true.");**

**System.*out*.println("The cell at (2,2) has a lizard (" + (game.getCell(2, 2).getLizard() != null) + "), expected true.");**

The **LizardGame** class models the game and its methods are typically called by the UI. Write simple tests to take place of the UI. The main interaction with the user happens when the user selects and drags the body segment of a lizard. These mouse events result in calls to the method **move()**.

Without a doubt, **move()** is the most complex of the methods you need to implement. Read the javadocs, review Figures 1 and 2 of this document, review the many methods available to help with the logic in **Lizard** and **LizardGame** and finally, break the problem into smaller parts and possibly create helper methods of you own. Many of the other methods in **LizardGame** are designed to help **move()** do its job, but you will also probably want to implement your own helper methods, which is allowed as long as they are **private**. Here are some simple tests for **LizardGame**.

**// Example tests for LizardGame**

**// (assuming previous tests worked and the game is loaded) segments = new ArrayList<BodySegment>();**

**segments.add(new BodySegment(liz, game.getCell(1, 0))); segments.add(new BodySegment(liz, game.getCell(2, 0))); segments.add(new BodySegment(liz, game.getCell(3, 0))); liz = new Lizard();**

**liz.setSegments(segments); System.*out*.println(); game.addLizard(liz);**

**System.*out*.println("Expected a message saying the number of lizards is now 2.");**

**System.*out*.println(); game.removeLizard(liz);**

**System.*out*.println("Expected a message saying the number of lizards is now 1.");**

**System.*out*.println();**

**liz = game.getLizards().get(0);**

**Cell adjCell = game.getAdjacentCell(1, 1, *RIGHT*);**

**System.*out*.println("Right of cell (1,1) is " + adjCell + ", expected cell (2,1,Ground,Empty)");**

**System.*out*.println("Cell (5,2) is available (" + game.getCell(5, 2) + "), expected true.");**

**System.*out*.println("Moving head of lizard one RIGHT."); game.move(4, 2, *RIGHT*);**

**System.*out*.println("Cell (5,2) is available (" + game.isAvailable(5, 2)**

**+ "), expected false.");**

**System.*out*.println("The head of the lizard is in cell (5,2) ("**

**+ (liz.getHeadSegment().getCell() == game.getCell(5, 2)) + "), expected true.");**

Do not use any outside libraries (i.e., external libraries such as Apache Commons; otherwise, we won’t be able to compile your code when grading.

# Style and documentation

The skeleton code has the public methods fully documented, there is not quite as much to do. Remember the following:

* Must add an @author tag with name to the javadoc at the top of each of the classes you write.
* Must javadoc each instance variable and helper method that is added. Anything added must be **private**.
* Since the code includes some potentially tricky loops to write, **add internal (//-style) comments, where appropriate**, to explain what you are doing inside the longer methods. Internal comments always *precede* the code they describe and are indented to the same level.
* Keep formatting clean and consistent.
* Avoid redundant instance variables.
* Accessor methods should not modify instance variables.