**Final Exam Project**

For the final exam project, you will develop a game application called “Hungry Squirrel”. In this game, there is a Squirrel in a maze looking for nuts. You will guide the squirrel to find and eat the nuts. There are two types of nuts available in the maze, Almonds and Peanuts. The squirrel gains nutritional points as it finds and eats the nuts. Once it finds all the nuts in the maze, the game is over. To implement this game, you ***shall*** define 8 classes: Maze class, Entity class, Squirrel class, Wall class, Nut class, Almond class, Peanut class, and HungrySquirrelGame class. Each class is described in details in the following sections.

The following class diagram shows the class hierarchy that must be implemented:

Entity



<impl>

Wall

Squirrel

Nut

Almond

Movable

Peanut

# In this document, you find each class described in details. There is enough information to help you implement this game. The class description does not provide all the details such as Constructors, setter/getter methods or toString method, you should add them as you see fit. You are not required to implement the class attributes and methods exactly as specified here; feel free to add/remove attributes or methods to match how you think each class should be implemented. *However, you must define each class keep the relationship as defined in the above class diagram, and must use the object oriented approach similar to what’s described in this document. Pay attention to the “shall” words in the document. You must make sure to implement the shalls defined in this document.*

# The Maze Class

You ***shall*** define the ***Maze*** class. The maze is a 20 x 50 matrix represented by a 2-dimensional **Entity** array (The **Entity** Class is defined in the next paragraph). Walls in the maze are represented by ‘\*’ and empty positions by blank spaces. The maze **shall** be read from a text file “Maze.txt” (The Maze.txt file is provided). A squirrel is able to move to empty spaces in the maze.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*\* \*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*\* \*\*\*\*\*\*\*\*\*\*\*

\* \*\*

\* \*\*

\* \*\*

\* \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* \*\*

\* \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* \*\*

\* \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* \*\*

\* \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* \*\*

\* \*\*

\* \*\*

\* \*\*

\*\*\*\*\* \*\*\*\*\*\*\*\*\*\*\*\*\*\* \*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\* \*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\* \*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

The ***Maze*** class defines the following attributes:

1. ***Max\_Maze\_Row***: This ***class variable*** is a constant variable that defines the maximum number of rows in the maze (it should be set to 20 rows).
2. ***Max Maze Column***: This ***class variable*** is a constant variable that defines the maximum number of columns in the maze (it should be set to 50 columns).
3. ***maze[][]*** : This ***class variable*** ***shall*** be defined as a 2-dimentional **Entity** array that contains the full maze and the entities.

The ***Maze*** class implements the following methods:

***public static void create(String filename)***

This method reads the file passed to the method (e.g. Maze.txt) and initializes the 2-dimentional array with the maze content provided in the file.

***public static void display()***

This method displays the maze structure and the containing entities.

***public static boolean available(int row, int col)***

This method takes a row and a column and determines if the location is a blank space. If it is, it returns true; otherwise, it returns false.

# Entity Class

You ***shall*** define an abstract “***Entity***” class. Three types of entities exist in the Maze: Nut, Squirrel, and Wall. Each entity has three attributes:

1. ***symbol***: This instance variable is a character symbol by which an entity is identified on the Maze. For example, a squirrel is represented by ‘@’. Each nut will be represented by the first character of its name (e.g. Almond will be represented by ‘A’).
2. ***row***: This instance variable is the row position of the entity in the maze.
3. ***column***: This instance variable is the column position of the entity in the maze.

The abstract ***Entity*** class contains an abstract method:

***public void create();***

The ***Entity*** class contains the following concrete method:

***public Entity put(int newRow, int newCol)***

This method puts an entity at location (newRow, newCol) in the maze. This method returns an object that was replaced in the maze. (hint: This can be useful when moving the squirrel and determining if the squirrel found a nut).

# Movable Interface

The ***Movable*** interface declares a single method and is implemented by entities that can move in the maze (i.e. Squirrel)

***void move(char direction);***

# Wall Class

You ***shall*** define the Wall Class. The Wall class is inherited from Entity super class.  Walls ***shall*** remain stationary and don’t move in the maze.

# Squirrel Class

You ***shall*** define the Squirrel class. Squirrel ***shall*** be represented by the “@” symbol in the maze. Squirrel ***shall*** be able to move up, down, left and right. The initial location of the squirrel ***shall*** be determined by the user. The program ***shall*** prompt the user to enter the starting row and column of the squirrel. The squirrel cannot move over a wall (i.e. ‘\*’). However, the squirrel can over move a nut. Once the squirrel moves over a nut, it eats the nut and collects points. Each type of nut carries different point.

The ***Squirrel*** class is inherited from the ***Entity*** abstract super class and it implements the ***Movable*** interface. The Squirrel class contains two attributes:

1. ***pointsCollected***: This attribute provides the total points a squirrel has accumulated when eating nuts.
2. ***totalNutsEaten***: This attribute provides the total number of nuts eaten thus far.

The Squirrel class defines the following methods:

***public void create()***

This method provides the implementation of the abstract method in the ***Entity*** superclass. This method prompts the user to enter the initial location of the squirrel in the maze. You have to make sure the location provided by the user is valid and available. If the user provides an invalid (e.g. row 200, column 200) or unavailable location where there is already a wall (‘\*’), the program must ask the user to input a new set of row and column. This continues until the user provides a position that squirrel can start from. Again, keep in mind that a squirrel cannot be placed where there is a wall ‘\*’.

***public void move(char direction)***

This method moves the squirrel one position to the direction specified.

# Nut Class

Nut ***shall*** be an abstract class that is inherited from Entity super class. There are two types of nuts available: Almond and Peanut. When a squirrel eats a nut, it gains points. A nut ***shall*** be removed from the maze once it is eaten. There ***shall*** be total of 5 nuts in the maze. The nuts locations ***shall*** be random and have to be placed in valid locations, meaning that a nut cannot be put in a position that is already occupied by a wall (\*), a squirrel (@) or a previously created nut. The type of nut (peanut or almond) is random based on 50% probability.

The ***Nut*** class defines the following attribute:

1. ***Total Nuts***: This class variable is a constant variable that represents the total number of nuts that will be created for this game (5 nuts).
2. ***NutritionPoints:*** This instance variable stores the nutrition points of a nut.
3. ***Name:*** This String instance variable is the name of the nut (“Almond” or “Peanut”)

The ***Nut*** class implements the following methods:

***void create()***

This method implements the abstract method in the ***Entity*** superclass. This method randomly generated the location of the nut. Keep in mind that a nut cannot be placed over a wall (\*), a squirrel (@) or a previously created nut. In other words, it can only be placed where there is a blank space.

# Almond Class

You ***shall*** define the Almond class. Almond is a type of a Nut hence inherited from the Nut super class. An almond ***shall*** be represented by the character symbol ‘A’ in the maze and carries 5 nutritional points.

The ***Almond*** class defines the following attribute:

1. ***Nutrition Points***: This class variable is a constant variable that represents the nutrition points an almond carries.

# Peanut Class

You ***shall*** define the Peanut class. Peanut is a type of a Nut hence inherited from the Nut super class. A peanut ***shall*** be represented by the character symbol ‘P’ in the maze and carries 10 nutritional points.

The ***Peanut*** class defines the following attribute:

1. ***Nutrition Points***: This class variable is a constant variable that represents the nutrition points a peanut carries.

# HungrySquirrelGame Class

The ***HungrySquirrelGame*** class defines the main method. The flow of the main method can be something like this:

1. Call the create method of the Maze class to create the maze.
2. Create a Squirrel object. This creates the squirrel and puts the squirrel in the maze based on the user input.
3. Instantiate an array of Nut objects and determine and create the type of nut (almond or peanut).
4. Display the maze with all the entities created.
5. Accept user input to move the squirrel.
6. For every move the full maze with all the entities should be displayed.
7. Every time the squirrel eats a nut, program prints the points collected for the new nut and total points collected thus far.

**!!! Squirrel ate Almond and gained 5 points (Total 15 points) !!!**

1. Once the squirrel collects all the nuts, a message must be displayed and the game is over.

**“Squirrel successfully collected all the nuts. Total points 30.”**

**“Thank you for playing this game”**

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*\* \*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*\* \*\*\*\*\*\*\*\*\*\*\*\*

\* \*\*

\* \*\*

\* \*\*

\* \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* \*\*

\* \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* \*\*

\* \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* \*\*

\* \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* \*\*

\* \*\*

\* \*\*

\* \*\*

\*\*\*\*\* \*\*\*\*\*\*\*\*\*\*\*\*\*\* \*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\* \*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\* \*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Enter the Squirrel position (row , column): 1,1

Position not available. Try again!

Enter the Squirrel position (row , column): 7,23

User input accepted.

Enter commands u,d,l,r to move Up, Down, Left, and Right:

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*\* \*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*\* \*\*\*\*\*\*\*\*\*\*\*\*

\* @ \*\*

\* \*\*

\* A P \*\*

\* \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* \*\*

\* \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* \*\*

\* P \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* \*\*

\* \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* \*\*

\* A \*\*

\* \*\*

\* \*\*

\*\*\*\*\* \*\*\*\*\*\*\*\*\*\*\*\*\*\* \*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\* \*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\* P \*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Enter command: d

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*\* \*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*\* \*\*\*\*\*\*\*\*\*\*\*\*

\* \*\*

\* @ \*\*

\* A P \*\*

\* \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* \*\*

\* \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* \*\*

\* P \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* \*\*

\* \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* \*\*

\* A \*\*

\* \*\*

\* \*\*

\*\*\*\*\* \*\*\*\*\*\*\*\*\*\*\*\*\*\* \*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\* \*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\* P \*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*\* \*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*\* \*\*\*\*\*\*\*\*\*\*\*\*

\* \*\*

\* \*\*

\* \*\*

\* \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* \*\*

\* \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* \*\*

\* \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* \*\*

\* \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* \*\*

\* \*\*

\* \*\*

\* \*\*

\*\*\*\*\* \*\*\*\*\*\*\*\*\*\*\*\*\*\* \*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\* \*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\* @ \*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Squirrel successfully collected all the nuts. Total points 40

Thank you for playing this game

# Extra Credit (+3 Points)

Define a new entity called “***PoisonousCashew***”. Poisonous cashews are bad for the squirrel and carry negative nutrition points (-15 points). If a squirrel eats a poisonous cashew and its total points become negative, the squirrel dies and the game is over. There are total of 5 poisonous cashews in the maze.

# Final Project Submittal Instructions and Grading Process

The final project must be submitted prior to midnight of the due date. Final project must be developed in NetBeans IDE. To submit your final project, you must create a zip file of your NetBeans project and upload the project on Canvas. The NetBeans project name should be your firstname and lastname with no spaces in between (e.g. AmirHallajpour) and the project zip file must have the same name.

I should be able to compile and run your project with no compile errors and run-time errors in the NetBeans IDE. ***If your project has any compile errors, that is considered an ‘F’ for the final grade***. ***Even if you don’t complete the project, it is very important that the completed portions of the project compile and run.***

**Good Luck**