

COMP3021: Java Programming (Fall 2016)

The Department of Computer Science and Engineering, HKUST

Programming Assignment #1: Text-based Pokémon

Contents

Background.....	1
Text-Based Pokémon.....	2
Specifications of the Input File	2
The Expected Output.....	4
Documentation using JavaDoc Syntax.....	4
Using OOP concepts	5
Marking Schemes	5
Input and Output file	6
Implementation Notes	7
Submission Details.....	9
Bonus.....	10
Deadline.....	11

Background

Pokémon Go is a free-to-play, location-based augmented reality game developed by Niantic for iOS, Android, and Apple Watch devices. It was initially released in July 2016 and got extremely popular worldwide. In this game, players can acquire random numbers of **Poke Balls** at different **Supply Stations**. When wild **Pokémon**s hidden in the surrounding environment are spotted, the players can use **Poke Balls** to catch these **Pokémon**s. The following shows some screenshots of this game.



Figure 1 Screenshots of Pokémon Go

In this assignment, you are required to implement a text-based Pokémon Go. We will give you a map with a number of Supply Stations and Pokémons. You are going to walk through the map from a point (denoted as starting point) to an end (denoted as destination point) by moving up, down, left and right. There may be several paths from the starting point to the destination, and you are required to output the optimum path under the defined scoring function.

Text-Based Pokémon

You are required to implement a text-based Pokémon in this assignment. The program will read all configuration from the input file to initialize the player, the Pokémon and the map. A single output file is created to show the path from the starting point to the destination along with the state of the player.

The input files:

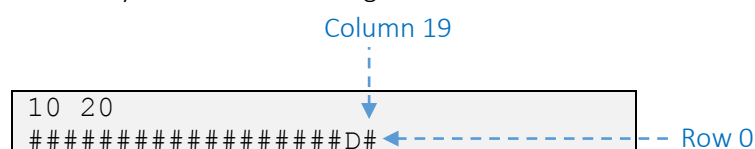
Descriptions of the input file
Describes the map
Describes the positions and properties of all Pokémons.
Describes the positions and configurations of all supply stations.

The output file:

Descriptions of the output file
Display the status of the player when he arrives at the destination.
Display the path the player has been visited

Specifications of the Input File

The input file, describes the map, the properties of Pokémons and Supply Stations. We are going to describe these aspects one by one as the following:



```

####          P      #
#### #####
#### #####
####P##### P
#### #####
#### #####
#### #####
B   S       P   S   #
##### ← Row 9
<1,14>, Spearow, Flying, 138, 4
<4,19>, Ninetales, Fire, 165, 8
<4,4>, Pidgey, Flying, 65, 3
<8,10>, Kakuna, Bug, 15, 1
<8,4>, 10
<8,14>, 15

```

The screenshot above shows an example of the input file. The first line contains two numbers **M** and **N**, which define the number of rows (i.e. 10 rows) and the number of columns (i.e. 20 columns), respectively. Both M and N are positive integers greater than zero. The following M lines define the map. Each line contains exactly N characters. Each cell <R,C> in the map is indexed by its row R and column C (Both R and C are indexed from 0). There are six different types of characters. “#”, “ ”, “B”, “D”, “S”, “P”. The meaning of these characters are shown as followings:

Character	Meaning	Can Pass or Not?
“#”	Wall cell, which any player cannot pass.	No
“ ”	Empty cell	Yes
“B”	The starting point of the player	Yes
“D”	The destination point of the player	Yes
“S”	Supply Station.	Yes
“P”	Pokémon	Yes

You (the Player) will **always** obtain all the Poke balls when you pass a supply station, and the Poke balls can only be obtained **only once** for each station. You will **always** catch the Pokémon when you pass it by default if you have enough Poke balls, and each Pokémon can only be caught **once**. If you do not have enough Poke balls when you pass a Pokémon, you cannot catch it at this time. However, you may catch it later when you have enough Poke balls.

This file then describes the properties of all Pokémon. Suppose there are **N_p** Pokémon in the map (**N_p** = 4 in the map shown above), there will be **N_p** lines after the map. Each line describes the position of the Pokémon (marked as <row number, column number>), the name of it, the type of species (Water, Bug, Flying, Ground, Poison, ...), the combat power of the Pokémon and the number of Poke Balls needed to catch it. These properties are separated by commas, and there **may** be extra spaces between them.

< Row Number, Column Number >, Name, Type, Combat Power, Number of Required Balls

Finally, the file describes the information of supply stations. Suppose there are **N_s** supply stations in the map (**N_s** = 2 in the map shown above), there will be **N_s** lines after the definition of Pokémon.

Each line describes the position of the supply station and how many Poke Balls you can obtain in this station. They are separated by commas and **may** contain spaces between them.

< Row Number, Column Number >, Number of Provided Balls

The Expected Output

There may be several paths from the starting point to the destination, and the player may pass different number of supply stations and catch different number of Pokémons. In this assignment, you are required to find a path that maximizes the following scoring function:

$$\text{scoring function} = < NB + 5 * NP + 10 * NS + MCP - Steps >$$

The table below shows the meanings of these symbol:

<i>NB</i>	<i><Number of Poke balls of the player when he arrives at the destination></i>
<i>NP</i>	<i><Number of Pokémons that the player has caught></i>
<i>NS</i>	<i><Number of distinct types of all Pokémons that the player has caught ></i>
<i>MCP</i>	<i><The maximum combat power of all Pokémons that the player has caught ></i>
<i>Steps</i>	<i><The steps of the move from the starting point to the destination></i>

The output file first prints out the value of the scoring function (denoted as VF) in the first line, it then prints out the final state of the player in the second line, including the following information:

NB: NP: NS: MCP

It then prints the path that the player has visited from the starting point to the destination with an arrow “->” between each step. All the steps should be outputted in **one** line. The following shows the output of the previous example:

```
187
13:3:3:165
<8,0>-><8,1>-><8,2>-><8,3>-><8,4>-><7,4>-><6,4>-><5,4>-><4,4>-><5,4>->
<6,4>-><7,4>-><8,4>-><8,5>-><8,6>-><8,7>-><8,8>-><8,9>-><8,10>-><8,11>
-><8,12>-><8,13>-><8,14>-><8,15>-><8,16>-><8,17>-><8,18>-><7,18>-><6,1
8>-><5,18>-><4,18>-><4,19>-><4,18>-><3,18>-><2,18>-><1,18>-><0,18>
```

In this example, the scoring function is evaluated to 187. The player obtained all the Poke balls at the two stations, and caught three Pokémons: Ninetales, Pidgey and Kakuna in total. In order to obtain all these things from the starting point to the destination, the player has moved 37 steps.

Documentation using JavaDoc Syntax

It is important for a software project to provide documentation on your code even though you are the sole programmer. JavaDoc is a standard documentation syntax defined in Java. In this programming assignment, you should write documentation to describe different classes and member functions you have created. Here is an example:

```

/**
 * Attempts to load an available buffer and blocks otherwise
 * @param buffer
 * @return
 * @tag usage.threading -id="4534354" : Do not invoke from UI thread
 * @tag task.todo -id="4534324" : Clear all uses of this method from UI thread
 */
public String getBufferWithPotentialBlocking(String buffer)
{
    String result = this.getBufferIfReadilyAvailable(buffer, false);
    if(result!=null)
        return result;

    // TODO: Synchronize

    String bufferId = getBufferIdFromBufferReferenceString(buffer);
    return this.forceBufferLoadBlocking(bufferId);
}

```

Figure 2 Example of Java Doc

Using OOP concepts

Object-oriented programming (OOP) involves programming using *encapsulation*, *inheritance*, *dynamic binding* and *polymorphism*. Here is an example shows the data encapsulation.

```

private String myField; // "private" means access to this is restricted

public String getMyField()
{
    //include validation, logic, logging or whatever you like here
    return this.myField;
}

public void setMyField(String value)
{
    //include more logic
    this.myField = value;
}

```

Figure 3 Example of Data Encapsulation

Marking Schemes

- Please observe the submission deadline, later submission will not be accepted or evaluated.
- This assignment must be finished individually; plagiarism is not allowed. Please refer to the plagiarism policy on the web page(<https://course.cse.ust.hk/comp3021/index.html#policy>)

Description	Percentage
<u>The correctness of the sample test cases.</u> The input file is given as <i>sampleIn.txt</i> , the correct output is given as <i>sampleOut.txt</i> (Note: Your output should be exactly matched with the sample output, please use the diff tool in Eclipse to check them carefully).	25%
<u>Documentation using JavaDoc.</u> You can get full mark if you write Java doc for all the defined classes and methods.	5%
<u>Using OOP concepts in your project.</u> You can get full mark if you have used both the concepts of Encapsulation and Inheritance .	5%
<u>Successfully declaring and defining the necessary classes.</u> (Game.java, Player.java, Station.java, Pokemon.java, Map.java, 4% for each class)	20%
<u>Other test cases</u> We will provide several other test cases with different maps.	45%

<p>For each of the test case, we will first check the output path is valid or not. A valid path satisfies the following criterions:</p> <ul style="list-style-type: none"> • Starts at the starting point and ends at the destination point. • No blocking walls in the middle. • Do not excel the border of the map. • The final state of the player you output is in consistent with the path, which means if the player walks along the path you output, the state of the player (i.e. VF, NB, NP, NS, MCP) should be exactly the same as what you output when he arrives at the destination point. <p>If the path is invalid, 0 credit will be obtained. Otherwise, we check the path is the optimum or not. You can get partial marks even if it is not the optimum. Suppose the credit for this test case is CT, the value of the scoring function you output is VF, the optimum is OVF, the credit you can obtain for this test case is $CT * VF/OVF$.</p> <p>The complexity of the other test cases is pretty much the same as the sample test cases which promises the following constraints:</p> $5 \leq M \leq 10; 10 \leq N \leq 30; N_p \leq 10; N_s \leq 5;$ <p>It's guaranteed that at least 20% of the cell are walls</p>	
--	--

Input and Output file

The sample input file (e.g. sampleIn.txt) is put in the same location of the **src** folder of your java project, and you are required to output the file (e.g. sampleOut.txt) file to the same location.

Name	Date modified	Type	Size
bin	9/15/2016 4:39 PM	File folder	
src	9/14/2016 10:26 PM	File folder	
.classpath	9/14/2016 10:26 PM	CLASSPATH File	1 KB
.project	9/14/2016 10:26 PM	PROJECT File	1 KB
README.md	9/14/2016 10:26 PM	MD File	1 KB
sampleIn.txt	9/14/2016 10:26 PM	TXT File	1 KB
sampleOut.txt	9/14/2016 10:26 PM	TXT File	1 KB

Figure 4 Location of the Sample Input and Output File

Your project should be capable of configuring the names of the input and output files. Therefore, we pass the names of the input/output files through running arguments. We have already provided the skeleton code of the entry main function in class file Game.java as shown in Figure 5. You are supposed to read the input from the file <inputFile>, and output the results to the file <outputFile>.

```

5 public class Game {
6
7     public static void main(String[] args) throws Exception{
8         File inputFile = new File("./sampleIn.txt");
9         File outputFile = new File("./sampleOut.txt");
10
11         if (args.length > 0) {
12             inputFile = new File(args[0]);
13         }
14
15         if (args.length > 1) {
16             outputFile = new File(args[1]);
17         }
18
19         // TO DO
20         // Read the configures of the map and pokemons from the file inputFile
21         // and output the results to the file outputFile
22     }
23 }
24

```

Figure 5 Main Function of the Whole Program

Implementation Notes

- You are NOT required to provide any graphics user interface in this GUI.
- Your project should be named with "Pokemon_PA1", and all your defined classes should be put in the default folder. Please **DO NOT** put your source codes under any *packages* defined by yourselves. The overview of your project should be look like as Figure 6.

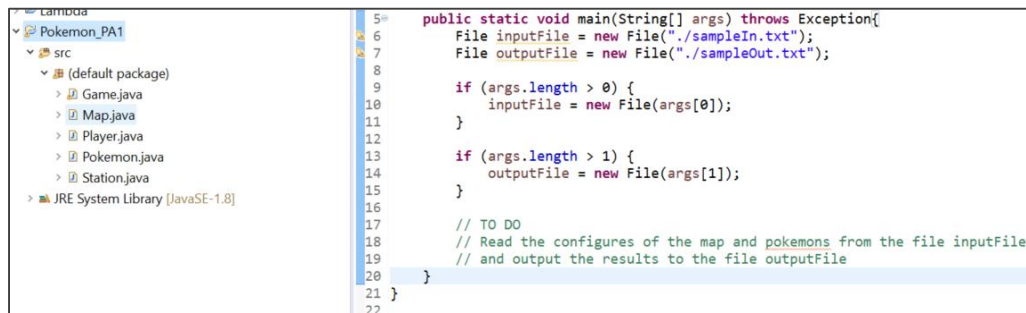


Figure 6 Project Overview

- The entry point of the whole project is the *main* function in the class **Game.java**, which is given by us. You are required to declare and define the following five classes:

Class Name	Class Description
Pokemon	Class Pokemon is a special type of cell. Besides recording the location of the cell, it also records the information of the properties of a Pokémon, including the name, type , combat power and required balls to catch it.
Station	Class Station is a special type of cell. Besides recording the location of the cell, it also records the information of a supply station, including the number of the provided balls.
Map	The class is defined to record the information of all the cells of the map, including walls, Pokemon cells, Station cells and empty cells.
Player	The class is used for recording the status of the player, including the current location, the caught Pokémons, the number of Poke balls and the path visited.
Game	This the entry class of the whole program, which contains the main function. It's also responsible for file input and output. It should contain a Map and a Player.

These classes are the required classes; you *may also need* to define *other* classes in order to finish this assignment.

- Tips on the searching Pokémons and the destination point. First, you need to find the starting point (which is denoted as B) from the map. In order to find a path to the destination, you need to keep trying moving up, down left and right. For example, if you are current at cell $\langle 2,3 \rangle$, you have four options for your next move as shown in the following picture:

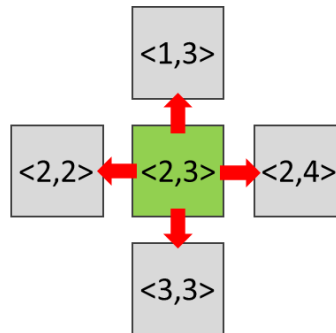


Figure 7 Moving Options

You need to try these four options one by one to see if you can make the move or not. You may not move to the next cell if it is a wall. After successfully moving to a new cell, you need to check whether it is the destination or not. If it is, you can print out the solution. You also need to check if the new cell is Supply Station or contains a Pokémon. If there are, you need to make the action accordingly: acquire the Poke balls or catch the Pokémon. If the cell is not the destination, you need to keep moving by adopting the same strategies.

In order to make your program more efficient and to avoid repeating visiting the same cell with the same status, you are suggested to store the status you have searched and avoid repeating the same status on the same cell.

- For all the test cases, we will test your program under the time budget of **1 minutes**, which means we will terminate your program if it runs over than **1 minutes**. In this case, you are suggested to output the results to the `<outputFile>` once you get a solution. Even if it is not the optimum, you can still get partial credits. If you have no output within **1 minutes**, you may not get any credit for this test case. (We will not test the program using complex maps, the constraints of the map are stated in the marking scheme).
- You can download a package provided from the Lab web page under the section of Assignment1 (<https://course.cse.ust.hk/comp3021/lab.html>), which contains the framework of source codes, the sample input/output and a runnable Pokemon_PA1.jar. The main entry function in Game.java is partially defined which has implemented the part of reading input/output filenames from system arguments. The rest parts remain to do. You are required to read configuration from the input file and write results to the output file. This package also contains a runnable jar file, which is a demo of the program you are going to implement. You can run this jar using command line. First open your terminal (terminal on Mac OS and Command Prompt on Windows) and go to the location where the jar file is located.

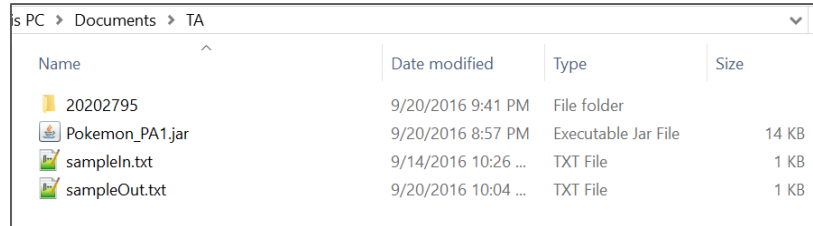
The screenshot shows a Windows File Explorer window with the address bar displaying 'PC > Documents > TA'. The window contains a table of files and folders.

Name	Date modified	Type	Size
20202795	9/20/2016 9:41 PM	File folder	
Pokemon_PA1.jar	9/20/2016 8:57 PM	Executable Jar File	14 KB
sampleIn.txt	9/14/2016 10:26 ...	TXT File	1 KB

Then, run the following command, which specifies the “sampleIn.txt” as the input file and “sampleOut.txt” as the output.

```
C:\Users\mwena\Documents\TA>java -jar Pokemon_PA1.jar sampleIn.txt sampleOut.txt
```

Wait for a while, you can see the output file under the same location:

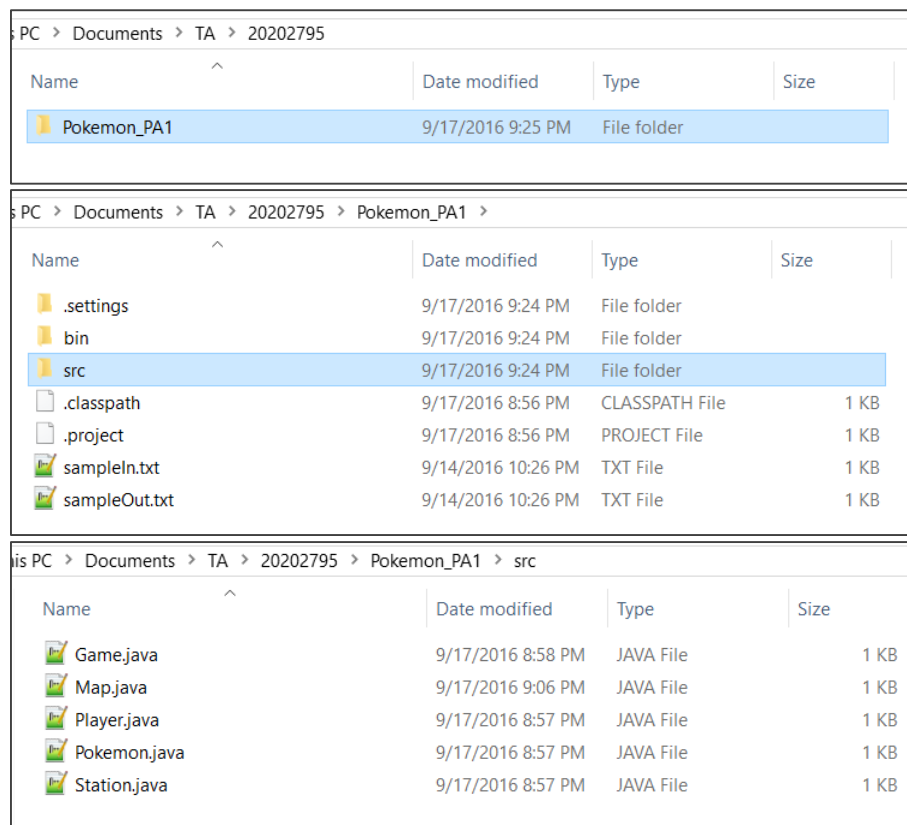


Name	Date modified	Type	Size
20202795	9/20/2016 9:41 PM	File folder	
Pokemon_PA1.jar	9/20/2016 8:57 PM	Executable Jar File	14 KB
sampleIn.txt	9/14/2016 10:26 ...	TXT File	1 KB
sampleOut.txt	9/20/2016 10:04 ...	TXT File	1 KB

You can compare the results generated by your program and the results generated by this demo program to guide your implementation of this assignment.

Submission Details

You are required to submit all your source codes through a zip file. Your file structure should be looks as following:



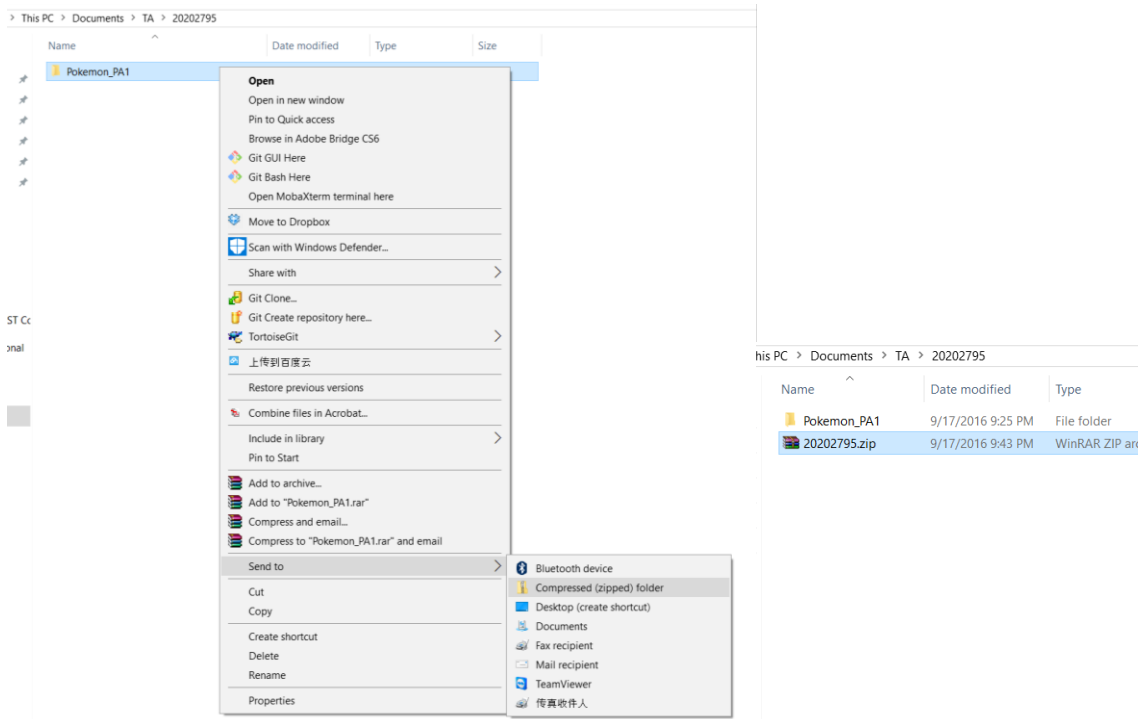
PC > Documents > TA > 20202795			
Name	Date modified	Type	Size
Pokemon_PA1	9/17/2016 9:25 PM	File folder	

PC > Documents > TA > 20202795 > Pokemon_PA1			
Name	Date modified	Type	Size
.settings	9/17/2016 9:24 PM	File folder	
bin	9/17/2016 9:24 PM	File folder	
src	9/17/2016 9:24 PM	File folder	
.classpath	9/17/2016 8:56 PM	CLASSPATH File	1 KB
.project	9/17/2016 8:56 PM	PROJECT File	1 KB
sampleIn.txt	9/14/2016 10:26 PM	TXT File	1 KB
sampleOut.txt	9/14/2016 10:26 PM	TXT File	1 KB

PC > Documents > TA > 20202795 > Pokemon_PA1 > src			
Name	Date modified	Type	Size
Game.java	9/17/2016 8:58 PM	JAVA File	1 KB
Map.java	9/17/2016 9:06 PM	JAVA File	1 KB
Player.java	9/17/2016 8:57 PM	JAVA File	1 KB
Pokemon.java	9/17/2016 8:57 PM	JAVA File	1 KB
Station.java	9/17/2016 8:57 PM	JAVA File	1 KB

Figure 8 File Structure

Please create the zip file by compressing your project folder “Pokemon_PA1”, and name the zip file with your student ID:



For Windows users, you can do it by right clicking the “Pokemon_PA1” folder, and choosing **Send to > Compressed(zipped) folder**. Please rename the archive file to <Your student ID>.zip.

For Mac users, you can do it by right clicking the folder “Pokemon_PA1”, and selecting **Compress items**, and then you can find the new created .zip archive in the same directory. Please rename the archive file to <Your student ID>.zip.

Please submit the zip file “<Your student Id>.zip” (i.e. 20202795.zip) via the CASS system (<https://course.cse.ust.hk/cass/student/#/submit>).

Bonus

You are welcome to propose additional interesting features in this assignment, and we can give a bonus of your new features up to **10%**.

To get this bonus, you need to submit a proposal of the design of your new features before **2016/10/06** (by sending us emails). Your proposed new features should be related to this assignment and can be incorporated into the assignment. You also need to specify what kinds of **new Java technologies** which are not included in this assignment (e.g. lambda expression, concurrency and so on) will be leveraged in order to implement your new features. Otherwise, your proposal may be rejected.

If your proposal is accepted, we will inform you the credit deserved for your proposed new features. If you have successfully implemented your new features based on your proposal in your final submission, you can get the bonus credit accordingly.

Deadline

The assignment will be due on: **2016/10/16**.

Submit a single file named as <Your Student Id>.zip.

Please follow the steps listed in the Submission Details.