CM10228 Coursework 3
Dungeon of Doom - Part 4

April 7, 2017

1 Introduction

Due Date: 19.00 on 5th May 2017

Overall, your mark in Programming 2 is composed of,

1. 50% coursework,
2. 50% exam.

The coursework component (part 1. above) is made up of three exercises (CW1, CW2 and CW3) each of which will build upon the last. These are as follows,

1. CW1: **Dungeon of Doom Part 2:**
   Extending code from CM102227 CW2 –or the supplied code- to allow multiple concurrent, networked agents in one dungeon (Java).
2. CW2: **Dungeon of Doom Part 3:** Adding a GUI (Java).
3. CW3: **Dungeon of Doom Part 4:** Reimplementing parts of the game in C.

This document provides requirements for the third coursework (CW3). CW3 is worth 2/5th of the coursework component (i.e. 20% of your total mark for the unit).

2 Coursework Specification

2.1 Core Requirements

This assignment asks you to make further extensions to the Dungeon of Doom game. More specifically, it asks you to write your own unit tests and reimplement parts of the game in C.

Note: it is highly recommended that you write your unit tests first before you reimplement functionality in C as this will help ensure that your new C code works as expected.

To meet the core requirements you should implement the following,

1. **Unit Tests for GameLogic and Map** - Create suitable unit tests for the GameLogic and Map classes including in the comments why these tests were chosen. For example, in the given code you should, as a minimum, include tests for the following methods,
   (a) GameLogic
      i. addDoDPlayer(int id)
      ii. getSpawnLocation()
      iii. processCommand(String action, int player)
      iv. move(DoDPlayer player, char direction)
      v. isAnotherPlayerOccupyingTile(int newX, int newY)
      vi. pickup(DoDPlayer player)
   (b) Map
      i. getMapName()
      ii. getMapWidth()
      iii. getMapHeight()
      iv. getGoldToWin()
      v. look(int x, int y)
      vi. getTile(int x, int y)
      vii. replaceTile(int x, int y, char with)
2. **Implement in C the functionality contained in the Map Java class EXCLUDING the functionality which loads a map from file** - That is,
   (a) create a new Java class called MapJNI which contains all of the methods in Map as native methods e.g.

public native char getTile(int x, int y);

NOTE: you should exclude the methods in Map which provide the functionality to load a map from a specified file i.e. in the given code you would exclude the methods *readMap(String fileName)* and *loadMap(BufferedReader reader)*

(b) use the JNI framework to generate a MapJNI.h file

(c) implement the functions in MapJNI.h in your own C file

(d) NOTE: values for the map, map name, map width, map height and win condition should be hard coded as global variables

(e) finally, replace in GameLogic the instantiation of Map with MapJNI and test to make sure your new C code works

You can choose to add this new functionality to 1. the code you submitted for CW1 or 2 in CM10228 or 2. to use the CW2 example code available on Moodle. NOTE: if you do choose to extend our example code you can still achieve the maximum marks possible for this coursework.

2.2 Core Requirements - Automated Testing

Important: The code you submit will be tested using a test suit which, will check your implementations by running both the DoD client and server. To allow us to run this test suit your submitted code **MUST** use the classes below which are included in our given code on Moodle.

1. **DODServerGUI:** - launches the server (and if included the server GUI) using the port number supplied as a default value.

   Usage: java DODServerGUI <portnumber>

2. **HumanClientGUI** - launches the human player client (and if included the client GUI) using the hostname and port number supplied as default values.

   Usage: java HumanClientGUI <hostname> <portnumber>

3. **BotClientGUI:** - launches the bot player client (and if included the bot GUI) using the hostname and port number supplied as default values.

   Usage: java BotClientGUI <hostname> <portnumber>

Failure to follow the submission specifications, would result a *-10* penalty applied to the final mark.

2.3 Advance Features

In order to potentially get full marks on the coursework you will also need to implement the advanced features below.

2.3.1 Implement the functionality which loads a map from file in C

1. that is, add, as appropriate, the remaining methods from the Map class as native methods in your MapJNI class
2. re-generate the MapJNI.h file
3. implement the functionality which loads a map from a specified file such that you can replace the hard coded variables for the map, map name, map width, map height and win condition

2.3.2 Implement GameLogic in C

That is,

1. create a new Java class called GameLogicJNI where the methods in GameLogic now call native methods rather then perform any functionality, e.g.

```
public synchronized String processCommand
    (String action, int player){
    processCommandJNI(action, player);
}

public native String processCommandJNI
    (String action, int player);
```

2. use the JNI framework to generate a GameLogicJNI.h file
3. implement the functions in GameLogicJNI.h in you own C file
4. finally, replace where appropriate the instantiation of the GameLogic Java class with GameLogicJNI and test to make sure your new C code works

2.4 One Final Suggestion

Note that we are not asking for any documentation of your requirements-gathering and design for this coursework, but you may find that doing these properly still helps you perform the tasks above, regardless of whether anyone looks at the output. You may also want to show these requirements and/or design documents to the tutors in lab in the early stages of development just to get feedback about whether you are on the right track.

3 Submission

By the date/time specified above, you should upload a zip file. The name of the zip file should be in the form:

CW3-<bucs username>.zip
e.g. CW3-abc123.zip

The zip file must contain a project folder, titled *Project*, containing your source code and any resources files needed. You should not include packages or other subfolders. No compiled code nor non-needed files, i.e. version control files, should be included.

4 Marking Scheme

Criteria	Max Score	Description
Core Specifications		
Unit Tests for GameLogic and Map	20	suitable unit tests for the GameLogic and Map classes, comments detail why tests were chosen
Implement Map in C EXCLUDING loading from file	20	Implements, in C, the functionality contained in the Map Java class EXCLUDING the functionality which loads a map from file
Commenting, OO design, procedural design, formatting and clarity	20	Code uses clean code practices, procedural and object-oriented programming techniques where appropriate, and is consistently commented.
Advance Features		
Implement loading from file functionality in C	20	Implement the functionality which loads a map from file in C
Implement the GameLogic class in C	20	Implements, in C, the functionality contained in the GameLogic Java class