CM10228 / Programming 1B:

Assignment 1 (2016): Networking and Multiple Clients in the Dungeon

Introduction:

Assignment Date: 18th February 2016

Due date: Friday, 10th March 2016

Overall, your mark in Programming 1B is composed of

a. 50% coursework,
b. 50% exam.

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

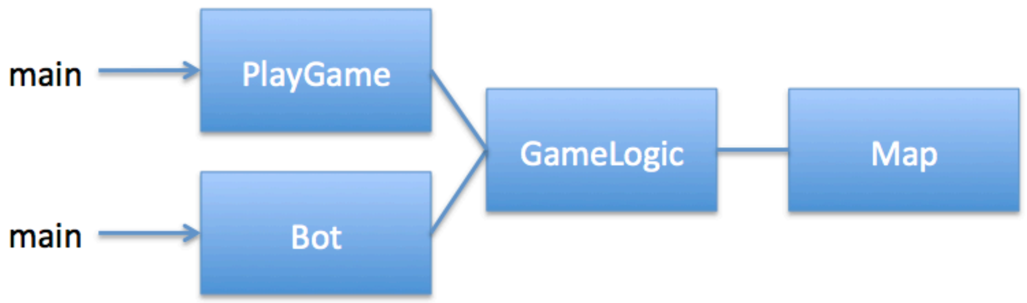
CW1.	Dungeon of Dooom: Extending prewritten code to allow multiple concurrent, networked agents in one dungeon (Java)
CW2.	Dungeon of Dooom: Adding a GUI (Java)
CW3.	Dungeon of Dooom: Extending game logic (C)

This document provides requirements for the first coursework (CW1). CW1 is worth 1/5th of the coursework component (i.e. 10% of your total mark for the unit). This means that it is worth less than CW2 and CW3 but should take you less time to complete.

What we have provided:

We have provided basic Java code that will, when compiled and executed, allow you to play a single player game call "Dungeons of Doom". The game "Dungeons of Dooom" is described in a separate document, which is also available on Moodle.

Our code is made up of four classes, organised as follows:



The Map class

- Reads one of the sample ASCII-art map files provided with the code on Moodle (and can, in fact, load any map written in the same format).
- Stores the map information in a 2D array (char[][]).

The GameLogic class

- Uses Map to load a map from file.
- Randomly positions a player within a map (on a non-wall space).
- Supports methods which fulfill the Dungeons of Dooom rules provided in the game description document introduced above.

The PlayGame

- Reads & writes to STDIN and STDOUT (the command line).
- Allows a user to control a player moving around the map held in the GameLogic class using the commands listed in the game rules (i.e. call the appropriate methods in GameLogic).

The Bot class creates a bot that

1. Can play the game (as a drop-in replacement for PlayGame, with its own main function).
2. Moves randomly.

The IGameLogic interface that

1. Hides the implementation detail of GameLogic
2. Defines the methods, which correspond to commands in the spec, which must be implemented to play Dungeon of Dooom

What we are asking you to do:

This coursework (CW1) emphasises networking, threading and concurrency. It also tests your ability to read and extend other people's code. To complete the coursework, you will need to extend our code to provide the following additional functionality:

1. Your version of Dungeon of Dooom should work on a client/server model.
   i.e. you should extend our code to allow the command line interface code (PlayGame, Bot) to connect to the game logic code (GameLogic, IGameLogic) over a network
2. Your code allows you to play Dungeon of Dooom against your bot in the same dungeon. You should also be able to support your friends and their bots joining you in the dungeon.

   Multiple bots should be able to move through the dungeon simultaneously (i.e. multiple instances of PlayGame should be able to connect to one instance of GameLogic).

3. No location in the dungeon can be occupied by two actors (i.e. human players or bots) simultaneously
   Note: our code does not check for this
4. Your bots are not so much smarter, faster & better than you that they are no fun to play against. Avoid walking through walls. Pick up any gold that it lands on.

   You will probably want to use sleep() commands in order to make the bot run at such a speed that you are able to interact with it (whether positively or negatively.) For your own personal satisfaction, you may want to have your bot learn (e.g. set its own delay between actions). You won't get any credit for this directly, but it might make your game more fun to play.

Note that we are not asking for any documentation of your requirements-gathering & design phase 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 a spec. to the tutors in lab in the early stages of development, just to get feedback about whether you are on the right track.

How to approach this coursework:

You may want to consider an approach to this coursework, in which you break the work to be done into a small number of sub-tasks or ‘steps’. This section describes one such breakdown.

Whilst we think that these steps will provide you with a useful path to completing the coursework, you are not obliged to follow any or all of them. You do, however, have to find some route to completion of the coursework objectives.

If, however, you do follow our suggested steps, please note that

- you would need to complete every step perfectly in order to achieve full marks
- *but* you can achieve a pass mark *without* completing every step.

If you gained excellent marks for the work described in steps 1-4 you could, theoretically gain more than 40% for CW1. It is safer, however, to go further than step 4, since you may lose marks on any or all of the early steps.

Our suggested breakdown of the coursework (i.e. the steps we suggest that you take) are as follows:

1 First, examine the code and make sure you understand it.

2 Separate out the code that you have been provided with into a server and a client. The networking code should be created in a class that is separate from the files you have been provided with.

- See the lecture notes on networking and talk to tutors in the labs. You are free to copy code from lecture notes, provided hat the source is properly acknowledged. Copying without giving acknowledgement is plagiarism and will be penalised if caught.

3 Change as little code for the networking as possible.

- Networking does not have to be hard, it is not that different from using streams like STDIN and STDOUT.

4 Create a Server class (Hint: by implementing IGameLogic).

- You probably want to use code for accepting multiple clients, but a single client is enough to pass the coursework.
- For now, the server should start the game when it starts up.
- Be sure to shut down the server cleanly when the game ends.

5 Create a client using PlayGame from the code distributed with this assignment. The client & server should communicate using the strings provided in the protocol document.

- Be sure to have the client also print things out to your terminal screen, so you can see what you are doing when you play the game.
- Hint: again look at the interface IGameLogic.

6 Create a ClientThread which modifies your existing Server class to accept multiple clients.

- Note that you will want to protect access to your map (and any other shared data). You can use synchronise for this purpose
- One client might want to look at the map while another is moving around on it, or picking up an item. These things shouldn't be able to happen at the same time!

7 Debug your new client & server.

- Be sure to do this in good time so you can take your code back to a lab if necessary & get help from the tutors.

If you get this far you have a good chance to pass the coursework; the remaining steps would then just be for better marks:

8 Write some test code to make sure that your client & server are doing what you think they should.

9 Make the bot that we have provided into a client too. This should be easy if you have already made your other client, as you can reuse the client code.

10 Modify your bot to make sure it isn't too powerful (see earlier in the spec).

11 Invite some of your course mates to try their clients on your dungeon, so you can be certain that you got the protocol right.

What to hand in:

Submit on Moodle a file named <yourUsername>-coursework1.zip

Important: The client and server code you submit will be tested using a test suit. That is, the test suit will check to see whether your server code can accept connections from our client code and whether your client code can connect to our server. To allow us to run this test suit your submitted code MUST use the following IP address and port number in both client and server code. A penalty of 5 marks deduction will be applied to code which has to be modified manually to allow the test suit to run.

IP address : localhost

Port Number : 40004

Note: For this coursework, you need to upload your extended code to Moodle (i.e. a combination of your code and, where needed for the game to run, ours). You should upload a zip file containing the uncompiled (.java) versions of your code by the date/time specified above.

Upload the file to Moodle by the time specified. Failure to name your submission file correctly will incur a penalty. Remember, marks are not given just because your code compiles and runs. It should run robustly, and it should be well commented and well documented.

How you will be marked:

This mark scheme for this coursework is summarised below.

Marks Table:

Criteria	Max Score
Code Legibility	10
Commenting	10
Networking (1 Client communicates with server)	20
Concurrency (More than 1 Clients communicate with 1 server)	20
Collision Avoidance	10
Protocol (Tutor can connect and use their own client)	10
AI (Bot moves at visible rate)	10
Bot and human players can play together on the same map (e.g. bot is not too fast or too smart)	10
Total	(minimum 0, max 100)

Mark Guidance

The below are a rough guide to how you might expect to be marked by an indicative description of what might achieve particular degree categories.

Threshold for Pass (40%+)

1 Basic networking functions: able to demo on lcpu connecting user client to server.

2 Allows game play with at least all of the commands provided by our code.

3 Code is properly formatted & commented.

Good Pass (~55%)

1 Meets all the criteria of a threshold pass.

2 Provides networked clients for both user and AI bots

3 Supports full communication protocol, i.e. passes full set of commands from client to server, even if bots or game ignores some commands.

4 Threading done correctly and safely.

Distinction (70%+)

1 All criteria of a good pass are met.

2 Submitted code clearly allows interaction between human user & bot.

3 Protocol is functional: All commands provided by protocol can be passed across a network and acted upon.

4 Tutor is able to connect an additional client with the standard protocol.

5 Code structure is clear & easily comprehensible in a quick overview.

Final Notes:

Whilst satisfying the high level requirements, above, please note:

- You are encouraged to use the eclipse IDE in programming and debugging your code. However, your code must run from the command line, and should work if you run it from two different computers. This is how we will be marking it. The code should be platform and operating-system independent. It should run in Java 1.7. To be safe, you should probably test that your code works on one of the BUCS servers (At time of writing LCPU has Java 1.7).
- This assignment is expected to take the average student (one who got about 58 in Programming I) about 15 hours. As mentioned in class, this is a double unit, so it is expected you spend at least 12 hours a week of self-study on the course, besides the two hours in lab and the two hours in lecture. We expect you will spend about 7.5 self-study hours a week for the next two weeks on this assignment, in addition to the two hours you have in lab. (Notice this still leaves you plenty of time for lecture preparation.) Because programming takes different people radically different amounts of time, some students will spend something more or less than 15 hours on this assignment.
- An important note on plagiarism: Plagiarism occurs any time that you use someone else's ideas without acknowledging them. Plagiarism is much, much worse than asking for help – in fact, asking for help isn't bad at all. If you get help on any part of your assignment you should acknowledge this in the comments. This includes asking coursemates, tutors, or other staff questions, and of course any code or ideas you get from books or the Internet. Say exactly where your ideas came from, and exactly how much of the code you hand in is your own.