COSC 2123/1285 Algorithms and Analysis
Semester 2, 2016

Assignment 2
Guess Who

Due date: 11:59pm Sunday, October 16th 2016 **Weight:** 15%

Pairs Assignment

1 Objectives

There are three key objectives for this project:

- Implement a Guess Who game.
- Design and implement Guess Who guessing algorithms.
- Have fun!

This assignment is designed to be completed in *groups of 2, or pairs*. We suggest that you work in pairs, but you may work individually.

2 Background

Guess Who is a classic two player game. The game consists of a set of characters/persons[1] with various attributes, e.g., hair or eye colour. Each player initially choose a person from this set. The aim of the game is each player takes turns to guess their opponent's chosen person. Each turn, a player can ask a question, such as, "does your person have green eyes?", which the opponent answers yes/no. From the answer, a player can eliminate the possible persons that their opponent have. The game ends when a player make a correct guess of the opponent's chosen person. See `https://en.wikipedia.org/wiki/Guess_Who%3F` for more details.

Traditionally, Guess Who is played between human players. In this assignment, your group will develop algorithms to automatically play Guess Who.

3 Tasks

The project is broken up into a number of tasks to help you progress. Task A is to design and develop the data structures to store game information and the players. Task B and C develops algorithms to play Guess Who, using your own data structures from task A. There is also a bonus task. It is based on implementing your own type of player to beat the ones from task A and B. For details on how each task will be assessed, please see "Assessment" section.

Task A: Design and Implement the Data Structures of a Guess Who Game (3 marks)

In this task, your group will design and implement the way a Guess Who game is stored and its information accessed. This includes loading of the game from game configuration files (see section "Details of Files" for details) and the storing of the chosen person in a player class.

[1] We will use person from hereonin to avoid confusion with character/letter.

Your design of these data structures may want to consider the way the Guess Who game is setup. Each player store their own copies of the game configuration information, which avoids forcing all players to store their game information the same way. This allows each group to go about their design in the way they consider appropriate, and even players from different groups to play against each other.

Task B: Implement Random Guessing Player (5 marks)

In this task, your group will implement a random guessing player for Guess Who. Each round, this type of player considers the remaining candidates that could be their opponent's chosen person. Each candidate has a set of attribute-value pairs associated with them. Let the union of the attribute-value pairs sets of all candidates be denoted by $\mathcal{S}$. Then this type of player will random select a pair (a,v) from $\mathcal{S}$, and then ask whether the chosen person has the value of v for attribute a. If the answer is yes, then eliminate all candidates who don't have value v for attribute a. If the answer is no, then eliminate all candidates that have the value v for attribute a. Eventually, this type of player should have one remaining candidate, then the player should guess this candidate as the opponent's chosen person.

This algorithm works, because the set of persons are all *unique*, as in there is one more more attribute-value that sets them apart from the other persons in the game. This means each question will at least eliminate one person per round, and should not remove opponent's chosen person, so eventually this should be the remaining person.

Task C: Implement Binary Search based Guessing Player (5 marks)

In this task, your group will implement a smarter type of player. This one is based on the decrease-and-conquer principle, similar to binary search. Similar to the random guessing player, each round this type of player also considers the attribute-value pairs of the remaining candidates. However, instead of randomly asking about an attribute-value pair, this type of player will try to ask about a pair that eliminates as close to half the candidates (might not be possible to get exactly half, but as close as possible). Again, when there is only one candidate left, this player should guess this candidate as the opponent's chosen person.

This algorithm works, as it also decrease the number of candidates each round, but should not eliminate the actual opponent's chosen person. However, it should be on average faster than random guessing player, as each time, the number of candidates is roughly halved. Recall for binary search, it has good worst case complexity because it halves the problem each iteration. For both Tasks B and C, we are in fact implementing a binary decision tree (the following is not essential to complete the assignment, but provides the rationale why this works - your lecturer will discuss this in class more), where a non-leaf node represent an attribute and value the player guesses, and each children represent further guessing possibilities based on no (one child) or yes (other child) answers. As each node in the tree represents a question, a path from root to leaf in the decision tree constitutes a guessing trace. If the decision tree has minimal height (which it will more likely have if we more or less half candidates each time), recall that such a tree has the best worst case possible, meaning on average, the length of the guessing trace should be shorter than the guessing trace of the random strategy, which can have long paths for some candidates.

Bonus Task: Designing and Implementing Customised Guessing Player (3 marks)

Note that the bonus task is deliberately designed to take more time for less marks than the other tasks. Only attempt this after completing Tasks A–C.

In this task, your group will design and implement an algorithm that can, on average, beat the random and binary-search guessing players. This task is deliberately open-ended, for you to explore,

consider and design better algorithms. The lecturer is more than happy to discuss ideas you have for this.

4 Details for all tasks

To help you get started and to provide a framework for testing, you are provided with skeleton code that implements some of the mechanics of the game. The main class (GuessWho) implements functionality of a two player Guess Who game, a method to log the game to check correctness and to parse parameters. The list of files provided are listed in Table 1.

file	**description**
`GuessWho.java`	Class implementing basic framework of the Guess Who game. *Do not modify useless have to.*
`Player.java`	Interface class for a player. *Do not modify this file.*
`RandomGuessPlayer.java`	Class implementing the random guessing player.
`BinSearchGuessPlayer.java`	Class implementing the binary search based guessing player.
`CustomGuessPlayer.java`	Class implementing the cusomised guessing player (bonus task).
`Guess.java`	Class implementing a 'guess'. *Do not modify this file.*

Table 1: Table of supplied Java files.

The framework provided implements a two player game. As explained earlier, the framework is designed such that each player is allowed to have their own way of representing the game. This allows your players to play against some of ours, or even other groups (given certain conditions are satisfied, please ask your lecturer first). Also, it defines how the players should interact. Examine GuessWho.java, particularly the code that iterates through the rounds. Note each player takes turn at making a guess via a Guess object, then the opponent answers, then this answer is passed back to the player. Examine the Guess class and see "Guess Structure" below to understand how a guess is implemented in this framework.

The framework also automatically logs the question-answer traces of the game. This allows us to evaluate if your players avoid asking redundant questions and the answers are correct (see "Assessment" section for more details).

Note, you may modify the GuessWho class, but we suggest you not to, as this contains the code for the game mechanics and the logging code and you do not want to break this. We also strongly suggest to avoid modifying the "Do not modify" ones, as they form the interface between players. You may add methods and java files, but it should be within the structure of the skeleton code, i.e., keep the same directory structure. Similar to assignment 1, this is to minimise compiling and running issues. However, you can change all the *Player.java files, including implementing/extending from a common player parent class. However, ultimately your *Player classes must implement the Player interface.

Note that the onus is on you to ensure correct compilation on the core teaching servers.

As a friendly reminder, remember how packages work and IDE like Eclipse will automatically add the package qualifiers to files created in their environments.

Guess structure

There are two types of guesses, one for asking if opponent's chosen player has a certain attribute-value pair, the other for asking if the chosen player is a certain person.

In the Guess class, there are three attributes, mType, mAttribute and mValue. When asking about attribute-value pair, set mType to Attribute and mAttribute and mValue to the asked attribute and value respectively. When asking about if the chosen player is someone, set mType to Person, mAttribute to "" (empty string) and mValue to the person's name. See the Guess class for more details.

For a player, when the answer(Guess guess) method is called, they should return true if the question/guess is true, and false otherwise. For the receiveAnswer(Guess guess, boolean answer) method, the player that was asking the question is updated with the answer, and should also return true if the game has finished (i.e., they guessed a person and the answer was true). For all other types of guesses and questions, that player should return false. See Player interface and Guess Who class for more details.

Compiling and Executing

To compile the files, run the following command from the root directory (the directory that GuessWho.java is in):

```
javac -cp .:jopt-simple-5.0.2.jar *.java
```

Note that for Windows machine, remember to replace ':' with ';' in the classpath.

To run the Guess Who framework:

`java -cp .:jopt-simple-5.0.2.jar GuessWho [-l <game log file>] <game configuration file> <chosen person file> <player 1 type> <player 2 type>`

where

- game log file: name of the file to write the log of the game.
- game configuration file: name of the file that contains the attributes, values and persons in the Guess Who game.
- chosen person file: name of the file that specifies which person that each player have chosen.
- player 1 type: specifies which type of player to use for first player, one of [random — binary — custom]. random is the random guessing player, binary is the binary-search based guessing player, and custom is the customised guessing player.
- player 2 type: specifies which type of player to use for second player, one of [random — binary — custom].

The jar file contains a library for reading command line options.
We next describe the contents of the game configuration and chosen person files.

4.1 Details of Files

Game configuration file

The game configuration files specifies all the set of attributes, values and persons in a Guess Who game. The file has the following format:

```
[attribute 1] [list of values it can take]
[attribute 2] [list of values it can take]
...
[attribute n] [list of values it can take]

list of persons in the game
```

Both attribute and it list of values are strings, separated by a space.

Each person in the person list have the following format:

```
[person name]
[attribute1] [value of attribute1]
[attribute2] [value of attribute2]
...
[last attribute] [value of last attribute]
```

An example game configuration file is as follows:

```
hairLength short medium long
glasses round square none
eyeColor black brown blue green

P1
hairLength long
glasses round
eyeColor brown

P2
hairLength short
glasses round
eyeColor black

P3
hairLength medium
glasses none
eyeColor blue
```

This specifies the following game:

- Three attributes, hairLength, glasses and eyeColour.
- attribute hairLength can take values {short, medium, long}.
- attribute glasses can take values {round, square, none}.
- attribute eyeColor can take values {black, brown, blue, green}.
- Three persons in this game:
  - Person "P1" has hairLength = long, glasses = round and eyeColour = brown.
  - Person "P2" has hairLength = short, glasses = round and eyeColour = black.
  - Person "P3" has hairLength = medium, glasses = none and eyeColour = blue.

Chosen person file

The chosen person file specifies the person chosen by each player. It is formated as follows:

```
[name of chosen person for player 1] [name of chosen person for player 2]
```

The names are separated by a space.
An example chosen person file is as follows:

```
P1 P3
```

This specifies that player 1 has chosen person P1, and player 2 has chosen person P3.

4.2 Clarification to Specifications

Please periodically check the assignment FAQ for further clarifications about specifications. In addition, the lecturer will go through different aspects of the assignment each week, so even if you cannot make it to the lectures, be sure to check the course material page on Blackboard to see if there are additional notes posted.

5 Assessment

The project will be marked out of 15 (with possible bonus marks of 3).

The assessment in this project will be broken down into a number of components. The following criteria will be considered when allocating marks. All evaluation will be done on the core teaching servers.

Note that for this and all implemented player algorithms, they should only ask *non-redundant* questions. A non-redundant question should eliminates one or more of the candidate persons after being asked. A redundant question is one that doesn't eliminate any candidates, e.g., ask about an attribute-value pair that none of the remaining candidate have. This is one way we can evaluate the correctness of your implementations. The other approach is to evaluate whether the answers your implement player replies is correct, e.g., if their chosen person has blue eyes, and the question/guess asked by opponent is whether the person has blue eyes, then the correct answer is true/yes, and incorrect answer is false/no.

Task A (**3/15**):

We will assess your classes for modelling a Guess Who game as follows:

1. Does the design hold all the necessary game information, i.e., at least the attributes and values in the game, the persons in the game, the chosen person in one or more player classes?
2. Is it a modular and understandable design?
3. Is it able to load the game configuration settings correctly?

Task B (**5/15**):

For this task, we will evaluate your player algorithm on whether:

1. It implements a random guessing strategy, as outlined in the specifications.
2. Produces a non-redundant guessing and correct answer trace.

Task C (**5/15**):

Similar to task B, for this task, we will evaluate your player algorithm on whether:

1. It implements a binar-search based guessing strategy, as outlined in the specifications.
2. Produces a non-redundant guessing and correct answer trace.
3. Additionally, over a number of games, does it on average, beat the random guessing player of task B, i.e., does it win more than it loses against the random guessing player?

Coding style and Commenting (**2/15**):

You will be evaluated on your level of commenting, readability and modularity. This should be at least at the level expected of a second year undergraduate student who has done some programming courses.

Bonus Task (**3/3**):

Similar to task C, for this task, we will evaluate your player algorithm on whether:

1. Produces a non-redundant guessing and correct answer trace.
2. For each of the players in task B and C and over a number of games, does the customised guessing player on average beat the other two players, i.e., does it win more than it loses against each of the other types of player?

5.1 Late Submissions

Late submissions will incur a *deduction of 1.5 marks per day or part of day late*. Please ensure your submission is correct (all files are there, compiles etc), resubmissions after the due date and time will be considered as late submissions. The core teaching servers and blackboard can be slow, so please ensure you have your assignments are done and submitted a little before the submission deadline to avoid submitting late.

6 Team Structure

This project is designed to be done in *pairs* (group of two). If you have difficulty in finding a partner, post on the discussion forum or contact your lecturer. If there are issues with work division and workload in your group, please contact your lecture as soon as possible.

In addition, please submit what percentage each partner made to the assignment (a contribution sheet will be made available for you to fill in), and submit this sheet in your submission. The contributions of your group should add up to 100%. If the contribution percentages are not 50-50, the partner with less than 50% will have their marks reduced. Let student A has contribution X%, and student B has contribution Y%, and $X > Y$. The group is given a group mark of M. Student A will get M for assignment 1, but student B will get $\frac{M}{\frac{X}{Y}}$.

7 Submission

The final submission will consist of:

- Your Java source code of your implementations. We will provide details closer to submission date.

Note: submission of the code will be done via Blackboard.

8 Plagiarism Policy

University Policy on Academic Honesty and Plagiarism: You are reminded that all submitted project work in this subject is to be the work of you and your partner. It should not be shared with other groups, nor should you elicit external tutoring services to do the assignment for you. Multiple automated similarity checking software will be used to compare submissions. It is University policy that cheating by students in any form is not permitted, and that work submitted for assessment purposes must be the independent work of the student(s) concerned. Plagiarism of any form will result in zero marks being given for this assessment, and can result in disciplinary action.

For more details, please see the policy at `http://www1.rmit.edu.au/students/academic-integrity`.

9 Getting Help

There are multiple venues to get help. There are weekly consultation hours (see Blackboard for time and location details). In addition, you are encouraged to discuss any issues you have with your Tutor or Lab Demonstrator. We will also be posting common questions on Blackboard and we encourage you to check and participate in the discussion forum on Blackboard. Although we encourage participation in the forums, please refrain from posting solutions.