University of Glasgow | School of Computing Science

Assessed Coursework

Course Name	IT/SD Masters Team Project – 'Top Trumps'			
Coursework Number	1			
Deadline	Time:	16:30	Date:	Mon 13 Feb 2017
% Contribution to final course mark	100%		Notional hours	100 hrs per person
Solo or Group ✓	Solo		Group	✓
Submission Instructions	See details under 'Submission'			
Who Will Mark This? ✓	Lecturer ✓	Tutor	Other	
Feedback Type? ✓	Written ✓	Oral	Both	
Individual or Generic? ✓	Generic	Individual ✓ (by group)	Both	
Other Feedback Notes				
Discussion in Class? ✓	Yes	No ✓		
Please note: This coursework cannot be re-done.				

Code of Assessment Rules for Coursework Submission

Deadlines for the submission of coursework which is to be formally assessed will be published in course documentation, and work which is submitted later than the deadline will be subject to penalty as set out below. The primary grade and secondary band awarded for coursework which is submitted after the published deadline will be calculated as follows:

(i) in respect of work submitted not more than five working days after the deadline
  a. the work will be assessed in the usual way;
  b. the primary grade and secondary band so determined will then be reduced by two secondary bands for each working day (or part of a working day) the work was submitted late.

(ii) work submitted more than five working days after the deadline will be awarded Grade H.

Penalties for late submission of coursework will not be imposed if good cause is established for the late submission. You should submit documents supporting good cause via MyCampus.

Penalty for non-adherence to Submission Instructions is 2 bands

You must complete an "Own Work" form via https://webapps.dcs.gla.ac.uk/ETHICS for all coursework UNLESS submitted via Moodle

Marking Criteria
See Marking Criteria Section

IT/SD Masters Team Project 2016-17

Top Trumps Java Game

This assignment has a weighting totalling 100% in your grade for the course. Further details are provided in 'Submission'. ***YOU MUST ONLY USE PROGRAMMING AND DATABASE TECHNIQUES TAUGHT IN Programming and Database Theory and Applications. Do not use JFileChooser: it is too time-consuming for assessment when testing your program!***

Context

Top Trumps is a simple card game in which decks of cards are based on a theme. For example race cars, dinosaurs, and even TV shows like 'The Simpsons'. Within a deck each card represents an entity within that topic (e.g. T-Rex for dinosaurs or Bart Simpson for the Simpsons). Within a deck each card has the same list of characteristics. For example dinosaurs can have a height, weight, length, ferocity, and intelligence. Each card has a value for each characteristic of the deck. The objective of the game is to 'trump' your opponent by selecting a category (e.g. intelligence) and having a "better" value for your card than the opponent does in their current card.

Gameplay is as follows:

There must be at least two players. The deck of cards is divided between the players. The first player takes their topmost card and selects a characteristic. The value of that characteristic is compared against the value for the same characteristic in the other players' top card. The player with the best value for that characteristic wins the round and the winner takes all the cards from that round (including their own) and places them at the back of their deck. If there is a draw the cards from the round are placed in a new communal pile and a new characteristic is selected by the same player from the next card. The winner then takes the cards from the round and any in the communal pile. The winner of a round maintains the choice of category until they lose, then the choice moves to the next player. Players lose the game when they have no cards left; the player left with all the cards is the winner of the game.

Functionality

You can make the following assumptions to simplify the implementation. Do not add additional unnecessary functionality. This is ‘gold-plating’ and will not result in a better grade. If in doubt, consult the course co-ordinator in the lab.

- there should be one human player and up to 4 computer players
- if a deck does not divide equally between the players, then some players may have less cards. For example if there are 3 players and 40 cards, then two players receive 13 cards and one player receives 14.
- a deck has 5 criteria and the criteria are always positive integers between 1 and 50 (inclusive)
- a higher number is always better for any given characteristic
- there are 40 cards in a deck.txt file.
- the first player should be selected at random
- a draw won't continue until the point where there are only cards in the communal pile – you do not need to deal with this programmatically, just assume that should this happen you are not expected to deal with it

Details of all cards in a deck should be held in a text file called '`deck.txt`'. The first line of the text file should contain a list of the categories in the deck, separated by a space. All decks should have a 'description' category to label the individual cards. A description within a deck can be assumed to be unique and a single word. The subsequent lines contain details of one card. You can assume the order of the categories from the first line align with the values provided for the cards and that all cards have a value for all categories in a deck. You can assume that all categories are single words.

For example in a dinosaur deck (numbers bear no resemblance to reality):

```
description height weight length ferocity intelligence
TRex 6 6 12 9 9
Stegosaurus 4 3 8 1 8
Brachiosaurus 12 8 16 2 6
Velociraptor 3 5 5 12 10
Carnotaurus 5 6 7 9 8
Iguanodon 2 2 3 1 9
Megalosaurus 9 9 8 6 9
Oviraptor 8 7 4 3 2
Parasaurolophus 7 7 1 3 4
Ornithomimus 10 9 8 7 5
Protoceratops 9 5 4 7 10
Riojasaurus 6 1 4 7 7
Saurolophus 7 1 10 7 8
Styracosaurus 7 3 4 1 1
Xiaosaurus 10 6 5 7 2
```

and so forth – a full sample deck is provided on Moodle.

The program must first load all card details from the deck and shuffle them (randomly order them). The program should then deal the cards between the players. The user should then be shown the detail from their top card (note there is no need to visualise this in a complex manner, the card details can be shown in text) and the first player randomly selected. If the player is the computer it should select a category for play, if the user is the first player they should be allowed to select a category to play the round. The game play should then proceed as detailed in the section Context.

Upon completion of the game you should be able to select an option to 'output game results to database'. This should write the following information about the game play to a database:

- How many draws were there?
- Who won the game?
- How many rounds were played in the game?
- How many rounds did each player win?

You should select one team member's database on the yacata server, from the Database Theory and Applications course, to write to. It is important you do not remove the username and password from the final code, as this allows us to test the software. You should also provide details of this database (username, database name and password) in the report.

There should be a button available, so long as a game isn't currently in progress, which allows the user to connect to the database and get information about the game play. This should include the following:

- Number of games played overall
- How many times the computer has won
- How many times the human has won
- The average number of draws
- The largest number of rounds played in a single game

These values should be calculated using SQL. The program should also allow the user to write this information out to a file.

Testing
In addition to the functionality described above, you should implement the following to allow for ease of assessment.

Your program should allow the printing of the cards in a deck. Using `System.out.println` your program should print the contents of the following, separated by a line containing dashes "----------" at the appropriate times as mentioned below:

- The contents of the complete deck once it has been read in and constructed
- The contents of the complete deck after it has been shuffled
- The contents of the user's deck and the computer's deck(s) once they have been allocated. Be sure to indicate which the user's deck is and which the computer's deck(s) is.
- The contents of the communal pile when cards are added or removed from it
- The contents of the current cards in play (the cards from the top of the user's deck and the computer's deck(s))
- The category selected and corresponding values when a user or computer selects a category
- The contents of each deck after a round
- The winner of the game

GUI
The GUI design is down to you, but in particular it should display the following:

- Upon loading the program, the user should be presented the option to view overall game statistics (as detailed previously), or play a single game.
- During game play, the GUI should display the contents of the user's top card and, when it is their turn, allow the user to select a category to play against the computer.
- The GUI should clearly indicate who's turn it currently is, and only allow the user to select a category when it is their turn
- Once the category has been selected for a round and the values compared, the GUI should display the values of the category for each player and highlight who won the round.
- If a draw should occur, the user should be notified of this.
- The GUI should contain an indication of how many cards are in the communal pile
- The GUI should contain an indication of how many cards are left in the user's deck and in the computer's deck(s)
- When the round played results in the game finishing, an indication of the overall winner should be presented to the user along with an option to update the database with the statistics of the game as previously described
- A screen should be accessible which shows the overall statistics and allow these values to be written to a file as described previously

Development
Working in a team of 4 or 5 students, you are required to design and implement a Java program with the above functionality, and with a suitable GUI.

Read this entire document carefully **before** starting work on the design and implementation of your program. This will ensure that you understand exactly what is required.

The project should be developed using Scrum, with 2 sprints each lasting 2 weeks. Your program design should use the MVC (Model–View–Controller) architecture, as explained in the *Programming* course.

Do not be tempted to add functionality that is not actually required (gold-plating). For example, you can assume that the input text-file is correctly formatted, so you need not waste time with data validation. You will not get a better grade for gold-plating.

Fine details of the GUI (colour scheme, font size, etc.) are not important. However, a GUI design that addresses usability issues, and how they impact on functionality, is important. Consider, for example, a GUI in which the text fields have no labels beside them; the program might work, and have all the required functionality, but it is not very usable, since the user has to guess what should be entered into each text field!

You will need to share code between members of your team. The *Software Engineering* course will likely introduce techniques for managing code, but you are not expected to use these techniques for this relatively small project. Just share code between team members, making sure that every team member has an up-to-date version.

If your team has difficulty meeting the deadline, it is better to submit a working program that omits some of the functionality, rather than a non-working program that attempts all the functionality.

After submission your program will be tested, so it is particularly important that your program does not refer to the absolute location of the deck data file. When testing your program within Eclipse, store the deck data file in the current working directory. Do not use `JFileChooser`: it is too time-consuming for assessment when testing your program! You should also develop in the default package – do not use your own package names.

As the final step of your development, you are required to create a jar-file named `TeamProj.jar` (using Eclipse, as explained in *Programming*). When the jar-file (or a shortcut) is placed on the desktop, the program can be run simply by double-clicking it.

Submission

Your team must submit a *single zip-file* containing the following items:

- your group report (see details below)
- your individual Java files
- your jar-file

Your zip-file must be named `TeamX.zip` (where X is replaced with your team letter or name). Each member must upload it on Moodle using the submission slot available along with their deltas. The final submission is due by **Monday 13 February 2017** at **16:30**

The University's standard penalty (2 bands per working day or part thereof, up to at most 5 working days) will be applied to any project submitted after the deadline.

Each member of the team must also submit by **Monday 13 February 2017** at **16:30 in the relevant Moodle submission slots**:

- an individual report in *a single .pdf with the name format surname_individualReport.pdf* where surname is replaced with your surname (see details below)
- a completed peer evaluation form (template available on Moodle) - if you fail to submit this then it will be assumed you distribute points equally between all team members

Group Report

Your report must concisely summarise the current status of your program. The report should consist of:

- **Cover sheet**: Show your team name, and the names and GUIDs of all team members.
- **Scanned copies of all of your user story cards, together with the estimated and actual time** (in story points) spent on each story and the story priority. Include stories that did not make it into the final product.
- **For each sprint: the planning and review reports** together with the planned and actual velocity.

- **The project burndown chart**
- **Assumptions**: State any assumptions you have made that have affected the implementation of your program.
- **Testing**: Summarise how you tested your program, giving examples of test cases.
- **Deficiencies**: State any known deficiencies, such as missing functionality or incorrect behaviour. Also suggest how you would fix these deficiencies.
- **Screenshots**: to demonstrate functionality of your GUI , included as an appendix and referenced in the main body of the report.

Your report should be a PDF document and have the name TeamXReport where X is replaced by your team letter or name.

Individual Report

Each team member should submit a personal reflection on using Scrum, which should include:

- personal and other team member roles
- lessons learned from estimating stories
- stories and other requirements that were missed at the start and added later
- any requirements that were hard to express in Scrum.

The report should be 12 point Times New Roman and be no longer than 3 pages.

Marking Criteria

Each team's project will be graded independently by two examiners, using the following criterion matrix:

Grade (band)	**Functionality** (weight 0.4)	**Design** (weight 0.3)	**Testing and documentation** (weight 0.2)
A (A1–A5) excellent	complete functionality	excellent program design and GUI design	very thorough testing; excellent report
B (B1–B3) very good	minor omissions	minor weaknesses in program design or GUI design	thorough testing; very good report
C (C1–C3) good	one major omission	major weakness in program design or GUI design	fairly thorough testing; good report
D (D1–D3) satisfactory	two major omissions	major weakness in program design and GUI design	modest testing; fair report
E (E1–E3) weak	several major omissions	widespread weaknesses in program design and GUI design	haphazard testing; weak report
F/G (F1–G2) poor	little or no functionality	unusable design	little or no testing; poor report

The individual report and other SPM components will be marked by the course co-ordinator for SPM.

Peer Evaluation of Contributions

The team grade will be adjusted for each individual on the basis of their contribution to the project. You have 100 points to distribute amongst your team, see full information on the peer evaluation template on Moodle. If all members submit equal points per member then the grade for the team will be the same as each individual's grade. If not, each individual's grade will be adjusted on the basis of the peer evaluation points received from each team member *as well as academic judgement based on interactions of the members during meetings in the lab* sessions with the team.