# IT/SD MASTERS TEAM PROJECT 2019-20

## Top Trumps Java Game

*This assignment has a weighting totalling 100% in your grade for the course. Further details are provided in 'Submission'. **You should use programming and database techniques taught in Programming and Database Theory and Applications.***

## 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.

## Aim

The aim is to build a computer program to allow a user to play top trumps against one or more AI opponents given a deck. The program should have two modes:

- ***Command Line Mode***: The game is played only through command line input and output. In this mode, only one game can be played at a time. This mode should be selected via a '-c' flag when starting the program:
  - `java -jar TopTrumps.jar -c`
- ***Online Mode***: The game should be hosted as a web service, comprised of a REST API that provides remote access to the core game functionality and one or more web-pages that enable a user to play the game. In this mode, multiple users should be able to play the game concurrently (e.g. in different Web browser tabs). This mode should be selected via a '-o' flag when starting the program:
  - `java -jar TopTrumps.jar -o`

# Functionality

**In both modes**, the program needs to:

- Enable a user to play a game of top trumps with a deck that was loaded in when the program started.
- Store the results of past games played in a database as well as visualize that information to the user on-demand.

**In command line mode only**:

- Write a 'test log' to file that contains snapshots of the program's state as it runs.

We discuss each of these requirements in more detail below:

## Playing Top Trumps

The program should implement the top trumps game as described in the *Context* section above. 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.

- There should be one human player and up to 4 computer players (AIs)
- 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

The deck we will be using is stored in a text file called '`StarCitizenDeck.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 3 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 – the star citizen-based deck is provided with the Template Package that can be downloaded from 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 is randomly selected. If the player is an AI player 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.

## Persistent Game Data

Upon completion of the game, the user should automatically 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 also be possible, so long as a game isn't currently in progress, for the user to connect to the database and get information about previous games. 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.

## Test Log

In addition to the functionality described above, you should implement the following to allow for program debugging when in *command line mode only*. When the program is started, if a '-t' flag is set on the command line, then the program should write out an extensive log of its operation to a 'toptrumps.log' file in the same directory as the program is run, e.g.:

- `java -jar TopTrumps.jar -c -t`

If a toptrumps.log file already exists, your program should overwrite that file. Your program should print the following information to that file, 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

## Command Line Mode

When started in command line mode, the program should ask the user whether they want to see the statistics of past games (see the Persistent Game Data section) or whether they want to play a game. The user's choice should be obtained from standard in (System.in). If they select to see statistics of past games, the program should print the associated statistics to standard out (System.out) and then ask the same question again. If they select to play a game, a game instance should be started, and the core game loop initialized. For each round, the round number, the name of the active player and the card drawn by the player should be printed to standard out. If the user is the active player, then it should ask the player to select a category, and obtain the user's choice from standard in, otherwise the AI player should select a category. The program should then print to standard out the selected category, who won (or whether it was a draw), the winning card and whether the player has been eliminated (they have no cards left). Rounds should continue to be played until a winner is determined (only one player has cards left). Once the user has been eliminated, the remaining rounds should be completed automatically (without user input). At the end of the game, the overall winner should be printed, and the Persistent Game statistics should be updated. The user should then be asked if they want to print the statistics of past games or play another game. An example of the command line output for the program is provided in the 'CLI.example.txt' file on Moodle.

## Online Mode

Online mode is an extension to the command line mode that provides a version of the game that users can play through their Web browser. A Web application has two main components. First, a back-end Application Programming Interface (API) that provides remote access to the game functionalities (e.g. starting a new game, drawing cards or getting players/cards for display). Second, one or more webpages that use the back-end API to enable the user to play the game. In this case, each web page should be comprised of HTML elements that are displayed and Javascript functions that connect to the API.

The webpage design is down to you, but it should display the following:

- Upon loading the web page, 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 they are the active player, 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 and the database should be updated with the statistics of the game as previously described

A video of an example website is provided in Moodle.

## Development

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

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 website design (colour scheme, font size, etc.) are not important. However, a design that addresses usability issues, and how they impact on functionality, is important. Consider, for example, a website 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, it is recommended that you use them.
- 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.
- **Teams are required to build on top of the provided Template Package during development.**

As the final step of your development, you are required to create a runnable jar-file named `TopTrumps.jar` (using Eclipse, as explained in *Programming*).

## Template Package

A software template package is provided that contains a program skeleton for teams to start with. All teams must use this template package as the basis for their implementation. The Template Package provides:

- **Maven** configuration, enabling automatic importing of libraries for creating web services
- **Basic reading of command line flags** allowing the user to select CLI or Online mode
- A **Web service** that can be run out-of-the box with **examples**

The Template Package uses the **Maven** software project management and comprehension tool to import other libraries it needs to run. In this case, it imports two main libraries:

- Dropwizard (http://www.dropwizard.io/1.2.2/docs/) for hosting the Web service

Instructions for compiling and running the Template Package are provided in the Launch Lecture Slides available on Moodle.

## 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 17 February 2020** 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 17th of February 2018** at **16:30 in the relevant Moodle submission slot (as a zip file)**:

- 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

The marks for the submitted project are divided as follows:

- **Requirements Capture and Software Design: 20%**
- **Command Line Version of the Software Product: 50%**
- **Online Version of the Software Product: 30%**

A group mark is generated and then adjusted based on individual contribution for each team member.