

---

---

# COMPSCI5074: MSc IT+ Team Project M

## *Project Report*

---

---

By:

——- SCHKEMBE CHORBA ——-

Nadezhda Dimitrova - 2172605D  
Ventsislav Antov - 2504299A  
Tereza Buckova - 2200528B  
Filip Marinov - 2175499M  
Gareth Sears - 2493194S



School of Computing Science  
UNIVERSITY OF GLASGOW

An assessed team project submitted to the University of Glasgow in accordance with the requirements for the degree SOFTWARE DEVELOPMENT (MSc) in the School of Computing Science. Supervised by:

DR RICHARD MCCREADIE

FEBRUARY 2020

# Table of Contents

<b>1</b>	<b>Project Setup</b>	<b>2</b>
1.1	Technologies . . . . .	2
1.2	SCRUM . . . . .	3
<b>2</b>	<b>User story cards</b>	<b>4</b>
2.1	Sprint 1 . . . . .	4
2.2	Sprint 2 . . . . .	9
<b>3</b>	<b>Sprint1</b>	<b>14</b>
3.1	Planning stage . . . . .	14
3.2	Review and Reflection . . . . .	16
<b>4</b>	<b>Sprint 2</b>	<b>17</b>
4.1	Planning Stage . . . . .	17
4.2	Review and Reflection . . . . .	18
<b>5</b>	<b>Burndown Chart</b>	<b>20</b>
<b>6</b>	<b>Testing</b>	<b>22</b>
6.1	The Testing and Quality Assurance Process . . . . .	22
6.2	Testing Conventions . . . . .	22
6.3	Implementation and Process Adjustments . . . . .	23
6.4	Evaluation and Suggestions for Process Improvement . . . . .	24
<b>7</b>	<b>Assumptions</b>	<b>25</b>
<b>8</b>	<b>Deficiencies</b>	<b>27</b>
<b>9</b>	<b>Conclusion</b>	<b>28</b>
	<b>Bibliography</b>	<b>29</b>
<b>A</b>	<b>Diagrams</b>	<b>31</b>
<b>B</b>	<b>Functionality</b>	<b>32</b>
<b>C</b>	<b>Screenshots</b>	<b>37</b>
<b>D</b>	<b>Tests</b>	<b>39</b>

# 1 | Project Setup

Prior to the beginning of the sprints we decided to allocate time for setting up the software environment and technologies for project management. This chapter would provide an overview of the technologies that were used, as well as a walk through our preparation for the sprints.

## 1.1 Technologies

The following technologies were used for project management, development, version control and technical documentation (See Figure 1.1 below).

- **Java 8** - programming language of choice for the command line version of the game and API [1].
- **PostgreSQL** - used as a relational database management system [2].
- **Dropwizard** - Java framework used for developing RESTful web services [3].
- **Maven** - build automation tool, used for managing dependencies and the source paths of the project [4].
- **Slack** - messaging platform for establishing team communication [5].
- **Scrum** - agile process framework used for managing the software development process and allowing for self-organisation within the team [6].
- **Trello** - web-based Kanban-style list-making application, deployed for creating and managing task boards with user stories, development tickets etc [7].
- **HTML** - markup language used in the development of the web app [8].
- **JavaScript** - used modern ES6+ Javascript to create and control dynamic web content [9].
- **Bootstrap 4** - HTML, CSS and JS framework for developing the web game [10].
- **GitHub** - used for hosting the software development version control, using git [11].
- **Mockito** - Java-based library for unit testing [12].
- **JUnit 5** - unit testing framework [13].
- **Jacoco** - test coverage generator [14]
- **Codacy** - for Automated code review [15]
- **L<sup>A</sup>T<sub>E</sub>X** - document preparation system used for producing this report [16].



Figure 1.1: Software tools deployed in the project.

## 1.2 SCRUM

Following the SCRUM methodology [6] and recommendation for separation of work, we decided to have **2 sprints of 10 days each**, allowing us to have 5 days for writing the report or 'runway'.

- Week 1 - familiarisation with requirements and project setup.
- Sprint 1 - *Command Line Game*
- Sprint 2 - *Web game*
- Last 5 days - report writing

For recording the progress of the team, we agreed on having  $\frac{1}{4}$  **ideal work day** as the unit for our User Story Points, where a single story is **capped at 8 points**. For prioritisation we used the **MoSCow technique** [17] with decreasing priority 1 – 4, where:

- 1 point - "*must have*"
- 2 points - "*should have*"
- 3 points - "*could have*"
- 4 points - "*won't have this time*".

## 2 | User story cards

### 2.1 Sprint 1

<p><b>1.</b></p> <hr/> <p><i>I can choose between web app mode or command line mode when starting the program.</i></p> <p>Priority: 1 Estimated Cost: 1 Actual Cost: 1</p>	<ul style="list-style-type: none"><li>• Using the <code>tag -o</code> will run an application server which can be accessed online via the browser.</li><li>• Using the <code>tag -c</code> will run the command line application, which will display the welcome logo "top trumps".</li></ul>
<p><b>2.</b></p> <hr/> <p><i>I can interact with the game in the command line and see the messages about the state of game.</i></p> <p>Priority: 2 Estimated Cost: 5 Actual Cost: 2</p>	<ul style="list-style-type: none"><li>• In the command line mode, when starting a new game, a number of player's selection menu needs to be shown where the user can select the player count</li></ul>
<p><b>3.</b></p> <hr/> <p><i>I can quit the game at any user prompt within the game.</i></p> <p>Priority: 3 Estimated Cost: 0.5 Actual Cost: 1</p>	<ul style="list-style-type: none"><li>• Any time the user is prompted for input, typing the word "quit" will close the CLI program, returning to the operating system.</li></ul>
<p><b>4.</b></p> <hr/> <p><i>I can decide between displaying statistics or run the game when the program loads.</i></p> <p>Priority: 1 Estimated Cost: 2 Actual Cost: 2</p>	<ul style="list-style-type: none"><li>• The first prompt the user is presented with displays two enumerated options, "see statistics" or "play game".</li><li>• Typing the corresponding number and pressing enter will show a title for the desired screen.</li><li>• The user will be unable to access any other mode or state if they input anything not corresponding to the enumerated values.</li></ul>

5.

---

*When I choose to see statistics, I can see relevant data for all the played games and after that I can choose to again, either display statistics or start playing game.*

Priority: 1  
Estimated Cost: 6  
Actual Cost: 8

- After the "statistics" title, the following statistics will be displayed:
  1. NUMBER OF GAMES PLAYED OVERALL
  2. HOW MANY TIMES THE COMPUTER HAS WON
  3. HOW MANY TIMES THE HUMAN HAS WON
  4. THE AVERAGE NUMBER OF DRAWS
  5. THE LARGEST NUMBER OF ROUNDS PLAYED IN A SINGLE GAME
- The user will be returned to the initial "statistics" / "play game" prompt from the beginning of the program after seeing this information.
- These statistics will be updated after playing a game.

6.

---

*I want to play against 4 AI players that choose a category if it is their turn.*

Priority: 2  
Estimated Cost: 5  
Actual Cost: 3

- An AI player will be able to select the highest valued attribute from their card if it is their turn.
- AIs will play their turn without user intervention.
- The game model will have 5 player entities, 4 AI players and one user.

7.

---

*I want to start every game with the same number of cards as other players, from a shuffled deck of 40 cards, with any remaining cards being placed in a communal pile.*

Priority: 1  
Estimated Cost: 6  
Actual Cost: 8

- The deck will have a different order from how it is initially read in.
- Every player will have the same deck size at the beginning of the game. This will be the floor of  $\frac{\#cards}{\#players}$ .
- There will be a communal pile which has  $\#cards \% \#players$ .

**8.**

---

*When it is my turn, I am able to choose a category from my top card.*

Priority: 1  
Estimated Cost: 5  
Actual Cost: 8

- The user will be shown their card when it is their turn.
- This card will have an enumerated list of its attributes.
- The user will be able to input a value which corresponds to the enumerated value of their desired attribute.
- Inputting an invalid value will re-prompt the user for a valid input to avoid the game state being disrupted.

**9.**

---

*I want the deck cards to have the same 5 criteria, which are integer values between 1-50 inclusive.*

Priority: 2  
Estimated Cost: 1  
Actual Cost: 4

- The card parser will read a deck file, creating cards with 5 attributes.
- The values will not be checked, as the specification guarantees the validity of the deck file.

**10.**

---

*The player with the highest score is the winner of the round and collects all the round cards, including any in the communal pile.*

Priority: 2  
Estimated Cost: 4  
Actual Cost: 4

- The communal pile is empty after a winner is declared.
- The winner's deck contains the cards which were in the communal pile.
- The winner's deck contains the last 'hand' of each player in the previous round.
- The winner's deck size matches its previous size + the total cards won (number of other players + number of cards in communal pile).
- The winner is the player with the highest valued attribute.

**11.**

*The winner of a round chooses category for the next round*

Priority: 2  
Estimated Cost: 1  
Actual Cost: 1

- If there is a winner, the active player in the following round is the same as that winner.

**12.**

*If there is a draw, the cards are placed in a communal pile.*

Priority: 2  
Estimated Cost: 2  
Actual Cost: 1

- If there is no winner, the communal pile contains the last 'hand' of each player in the previous round.
- For each consecutive draw, the communal pile's size equals its previous size plus the number of players in the most recent round.

**13.**

*Players are eliminated from the game if they have no cards left.*

Priority: 2  
Estimated Cost: 1  
Actual Cost: 1

- For each player with no cards, the next round's players do not include those players.
- The remaining players' cards should add up to the original deck size.

**14.**

*If I am eliminated, I want to see how the game played out with the remaining rounds completed automatically.*

Priority: 2  
Estimated Cost: 2  
Actual Cost: 3

- No more input will be required from the user. The next prompt will be the original menu prompt.
- The results of each round will be displayed until a winner is declared.

**15.**

*At the end of the round, I want to see the winning selected category, who won, or if it was a draw. The winning card and if a player was eliminated.*

Priority: 2  
Estimated Cost: 2  
Actual Cost: 4

- At the end of a round, I will see titles and values in the CLI which correspond to the information on the card.



**16.**



*I want the statistics for every game recorded.*

Priority: 2  
Estimated Cost: 6  
Actual Cost: 6

- After a game finishes, the database will be updated with the game values.
- The next time the user sees statistics, they will be updated with the previous game's results.

**17.**



*After a game is over, I can choose to play again or see game statistics.*

Priority: 1  
Estimated Cost: 1  
Actual Cost: 1

- When a game finishes, the user will see the options and will be given a prompt which matches that found when the program is started.

## 2.2 Sprint 2

<p><b>18.</b></p> <hr/> <p><i>When I open the homepage I want to see a game and a statistics button and the Top Trumps title.</i></p> <p>Priority: 1 Estimated Cost: 2 Actual Cost: 2</p>	<ul style="list-style-type: none"><li>• When the homepage is loaded it needs to display buttons and top trumps title.</li><li>• The statistics button takes you to the statistics page.</li><li>• The game button takes you to the game page.</li></ul>
<p><b>19.</b></p> <hr/> <p><i>In the navigation bar I want to have a home button on top of the game page and the statistics page.</i></p> <p>Priority: 2 Estimated Cost: 1 Actual Cost: 1</p>	<ul style="list-style-type: none"><li>• When a game/statistics/homepage is loaded, there needs to be the same home button displayed across all three pages.</li><li>• The button takes you to the homepage.</li></ul>
<p><b>20.</b></p> <hr/> <p><i>When I open the statistics page I want to see the five overall game stats as a list.</i></p> <p>Priority: 1 Estimated Cost: 2 Actual Cost: 3</p>	<ul style="list-style-type: none"><li>• The statistics page contains 5 stats – the number of human wins, AI wins, longest game, average number of draws and games played in total, and corresponding numbers.</li><li>• These numbers are updated every time a game is finished and change accordingly.</li></ul>
<p><b>21.</b></p> <hr/> <p><i>I am able to play the game in multiple windows.</i></p> <p>Priority: 3 Estimated Cost: 8 Actual Cost: not implemented</p>	<ul style="list-style-type: none"><li>• When the game is opened in multiple browser windows, the user is able to start a new game session in each one of them.</li></ul>

<p><b>22.</b></p> <hr/> <p><i>When I go to the game page, a pop-up screen is displayed where I can select the number of opponents and start the game.</i></p> <p>Priority: 1 Estimated Cost: 2 Actual Cost: 3</p>	<ul style="list-style-type: none"> <li>• When the game page loads, a modal pops-up where the user is able to choose from 1-4 AI opponents.</li> <li>• It is not possible to start a game without selecting the number of opponents.</li> <li>• It is not possible to click out of the modal to close it.</li> <li>• Start button starts the game when clicked and the close button redirects to the homepage when clicked.</li> </ul>
<p><b>23.</b></p> <hr/> <p><i>When I initiate a game and select the number of opponents, I can see all of the players with their decks.</i></p> <p>Priority: 1 Estimated Cost: 3 Actual Cost: 4</p>	<ul style="list-style-type: none"> <li>• User can see the correct number of players and their decks displayed on the screen.</li> </ul>
<p><b>24.</b></p> <hr/> <p><i>On initialising a round, I can always see my top card. The cards of all the other player's are hidden.</i></p> <p>Priority: 1 Estimated Cost: 2 Actual Cost: 2</p>	<ul style="list-style-type: none"> <li>• When game is initialized the user's card with the card name and attributes is displayed.</li> <li>• All the other cards belonging to AI players will be hidden by a block of colour.</li> </ul>
<p><b>25.</b></p> <hr/> <p><i>I can see who the active player and winner of the round are.</i></p> <p>Priority: 2 Estimated Cost: 2 Actual Cost: 4</p>	<ul style="list-style-type: none"> <li>• The active player that chooses the attribute next is highlighted by green in their header.</li> <li>• The winner of the round has a crown icon in their header.</li> </ul>

<p><b>26.</b></p> <hr/> <p><i>Above each player's card there is a header with their name and the number of remaining cards.</i></p> <p>Priority: 2 Estimated Cost: 2 Actual Cost: 2</p>	<ul style="list-style-type: none"> <li>• The user will see piles for all players in the game and the deck will show how many cards are remaining.</li> <li>• The number on the decks will change when a round is played.</li> </ul>
<p><b>27.</b></p> <hr/> <p><i>I can see the chosen attribute on all player's cards after the round is played.</i></p> <p>Priority: 3 Estimated Cost: 2 Actual Cost: 2</p>	<ul style="list-style-type: none"> <li>• After a round is played, the user will see the attributes and associated values on cards for all players. The attribute that was played in the round will be highlighted on all cards.</li> </ul>
<p><b>28.</b></p> <hr/> <p><i>I can see the number of cards in the communal pile during the whole game.</i></p> <p>Priority: 2 Estimated Cost: 2 Actual Cost: 2</p>	<ul style="list-style-type: none"> <li>• The user will see a number of cards in the communal pile and this number can change.</li> <li>• If someone wins in a round then the number of cards in the communal pile drops to zero for the next round.</li> <li>• When there is a draw the number of cards in the communal pile increases by the number of the players.</li> </ul>
<p><b>29.</b></p> <hr/> <p><i>I can always see the round number during the game.</i></p> <p>Priority: 1 Estimated Cost: 0.5 Actual Cost: 1</p>	<ul style="list-style-type: none"> <li>• The user can see the round number displayed and it increases by one every played round.</li> <li>• The displayed round number is "1" at the beginning of a game.</li> </ul>

**30.**

---

*I can always see the round button before the round.*

Priority: 1  
Estimated Cost: 2  
Actual Cost: 2

- When it is the user's turn after they choose, the user can see the round button with a drop-down menu.
- If it is the AI's turn, the play round button changes into the next round button after a round is played.
- At the beginning of a round, the play round button is displayed again.

**31.**

---

*I can see when any player is eliminated.*

Priority: 1  
Estimated Cost: 2  
Actual Cost: 2

- When the player is eliminated, the user sees a message that the player is out and their name is highlighted in red.

**32.**

---

*When the game ends I can see whether the game was auto-completed, who won and I am able to either start a new game or see the overall statistics.*

Priority: 1  
Estimated Cost: 2  
Actual Cost: 3

- The user can see the name of the correct game winner displayed in a pop-up modal.
- The statistics button takes the player to see game stats and the new game button starts a new session.

**33.**

---

*If it is my turn, I can see a play round button with a drop-down menu. I can expand and select the attribute for the round.*

Priority: 1  
Estimated Cost: 2  
Actual Cost: 4

- When the user is an active player, the round button turns into a drop-down menu button with attributes in it.
- There are 5 attributes on the button that correspond to the attributes on the player's card.
- After clicking the button, the user plays the round with this selected attribute.

**34.**

*I can see a message after every round with the winning attribute and the player who won. If there is a draw, the message board indicates this.*

Priority: 2

Estimated Cost: 2

Actual Cost: 2

- The user can see a message board that shows the name of the active player, state of the game and the selected attribute for the round.
- The message changes every round.

## 3 | Sprint1

### 3.1 Planning stage

In this section we will cover the codebase and MVC architecture, class structure, tasks and processes that were discussed and agreed during the planning stage.

#### GitHub setup

When setting up GitHub, we decided to make use of *Codacy* for automated code reviews [15], to enforce quality standards and save time in reviews. Furthermore, nothing could be merged into `master` without a peer review. Ideally, we also agreed that tests should be passing successfully, before pushing. To support the parallel development between different members we agreed to use *supportive branches* [18], such as *feature*, *bug* and *hotfix*. Appropriate name conventions were to be followed, in the form of `feature/<td name>`.

#### Directory structure

This project implements a normal Maven-style directory structure (See Figure 3.1), that ensures test and source code are separated.

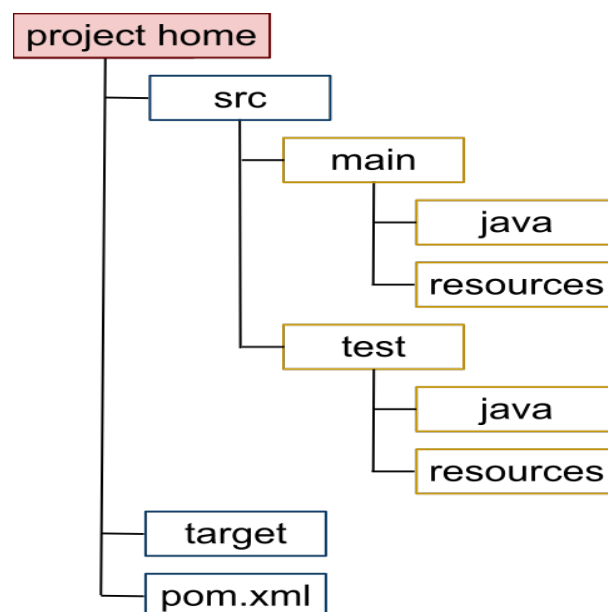


Figure 3.1: Maven-style directory structure for a Java project. Image taken from [19].

#### MVC Architecture

To design the MVC architecture for the project, we had to decide if the game loop will be in the model or the controller, how to separate view from controller, who handles the user input etc. We agreed that ideally we would want the model to be reusable for both CLI version of the game and the web app and we arrived at the following separation:

- **view** - user input encapsulated with the input stream returned to the controller as a String.

- `model` - contains the game logic, shared for both CLI and online game mode.
- `controller` - contains the game loop. API in the online mode.

### Class Structure

The separation of concerns was discussed during the sprint planning and the main classes were established:

- `Attribute` - will contain a name and value. For Example: "Strength": 6.
- `Card` - will have a name, as well as a list of `Attribute` Objects.
- `Pile` - to be implemented as a list of `Card` objects. In this way `Pile` is versatile and can serve as the initial whole deck, the player decks and the communal pile.
- `Player` - will contain a name, rounds won and a `Pile` to represent the player's deck. `AIPlayer` would extend `Player`.
- `Database` - a class that would be able to establish or close a connection to the PostgreSQL database, upload and retrieve game statistics.
- `GameModel` - as discussed previously
- `View` - as discussed previously
- `CLIcontroller` - as discussed previously

### Sequence diagrams

The team made two diagrams to represent the game flow from the user perspective, as well as a message-based round structure. Both of those would ensure a team-wide shared understanding of what needs to happen and when during each round and the overall game (See Appendix A). *Note:* Those diagrams are as constructed in Sprint 1 and the implementation since has changed.



### 3.2 Review and Reflection

Table 3.1: *End of Sprint 1 retrospective.*

END OF SPRINT RETROSPECTIVE	
Gone well	Gone not so well
✓ Successful completion of the main sprint objective: CLI game works with no known bugs or errors	✗ Logger has base functionality but no integration. Needs to roll over to the second sprint.
✓ Steep learning curve. Experienced working in an Agile team, following the Scrum methodology and produced a working prototype.	✗ Developers ended up focusing on stand alone classes, rather than proper user stories resulting in the need to "glue" things together and consolidate work and have a large group session for connecting the MVC, add functionalities, rename and bugfix.
✓ Automated coverage generally good	✗ Writing tests took longer than anticipated. Unable to write integration tests while waiting on other classes.
✓ Frequent meetings with excellent participation and contribution from all members.	✗ Delays in pull requests, reviews and branching too much caused slow progress and confusion.

The successes and shortcomings of this sprint were taken into consideration when planning for the next.

#### Actual and planned velocity

The user stories with acceptance tests were created and 50,5 points were planned for Sprint 1. The actual velocity of Sprint 1 was 58 story points.

## 4 | Sprint 2

### 4.1 Planning Stage

This section is based on the lessons learned from Sprint 1's review and will similarly cover areas such as the codebase, class structure, tasks and processes.

#### **New Approach**

Following the challenges experienced in Sprint 1 due to the allocation of the tasks to team members on a class-by-class basis, the team decided to focus more on functionality when approaching task distribution. In order to achieve this, the whole team agreed on a shared vision of the product wireframes based on the desired user stories allocated to the sprint. This was then proceeded by a discussion on the separate components necessary for answering questions around round initialization, API calling, information passed through the API, etc. In order to achieve optimal effectiveness, the team was split into two smaller teams. One focused on the front-end, and the second - on the back-end. Furthermore, the team decided to start the documentation for the project at the end of Sprint 1, so that the reporting is smoother and more accurate instead of recollecting details at the very end of the project. This was maintained during Sprint 2 as well.

#### **Technology Used**

The team had to adapt to using different tools quickly despite the fact that the majority of the members had no experience using some of the technology necessary for the completion of this part of the project. The tools mentioned include JavaScript ES6, Bootstrap and CSS.

#### **API and Frontend Components**

One of the API concerns to address was how to pass attributes from the AI players. There was a team discussion around how to structure the API so that it could be used more easily by the front-end team. This included serving players in a suitable object structure to allow the front-end to consume it within their own implementation of javascript view components. `PlayerFactory.js` is a good demonstration of this process. It uses the API response to populate view templates and exploits closures to provide a restricted interface for altering the view. This ultimately led to better code reuse and a reduced dependency on javascript global variables in the game page, although their current implementation does rely on relatively expensive DOM manipulation which could be optimised further.

#### **MVP**

The team decided to maintain focus on the MVP (Minimum Viable Product), and session management was left to work on after everything else was running. At some point the team agreed that the estimated story points for sessions would require to take it into a next sprint, given the available resources for the current sprint encapsulating all five members. This would have been impossible to finish within the time frame of the project including proper documentation. As a result, the team prioritised producing a clean and well documented code, with extensive testing and good write-up, as opposed to sacrificing those for the benefit of adding sessions.

## Functionality

The functionality of the online version of the game involved a home screen, a radio button pop-up for player number selection, game screen adapted to different scenarios for rounds, a game over pop-up and a statistics screen. All of these are illustrated in the appendix.

## 4.2 Review and Reflection

Table 4.1: *End of Sprint 2 retrospective.*

END OF SPRINT RETROSPECTIVE	
Gone well	Gone not so well
✓ Successful completion of the main sprint objective: MVP online game works with no known bugs or errors	✗ Session management not completed. Could potentially roll over to a third sprint if there was one.
✓ Much smoother and consistent work compared to Sprint 1. Reduced the "work-in-progress" cards, which is also visible from the Burndown chart.	✗ In the future, ideally the team would migrate from <code>ftl</code> and use a more contemporary framework because the tools that were used were outdated.
✓ Very well managed split of tasks between front-end and back-end by forming separate small teams to deal with each. Tasks were distributed based on competencies as well as preferences, which made the team overall satisfied and motivated.	✗ On the front-end side, the components that were made do not exactly follow a logical encapsulation pattern even though they manage to reduce the code complexity on the game page.
✓ Maintained frequent meetings, communication and participation from all members.	✗ Delays in pull requests, reviews and branching too much caused slow progress and confusion.
✓ Communication between the two teams was optimal and it allowed people to focus on learning one new technology at a time.	✗ Found cross-platform issues related to new line symbols between Mac and Windows. To solve this, the team removed all backslash <code>r</code> characters. In the future the team will have to make tests platform agnostic, instead of the current fix.

### **Actual and planned velocity**

Following the creation of user stories, 46,5 points were planned for Sprint 2 with actual velocity of 39 story points. There was 1 user story left with 8 story points estimated which belonged to the aforementioned multiple windows feature.

## 5 | Burndown Chart

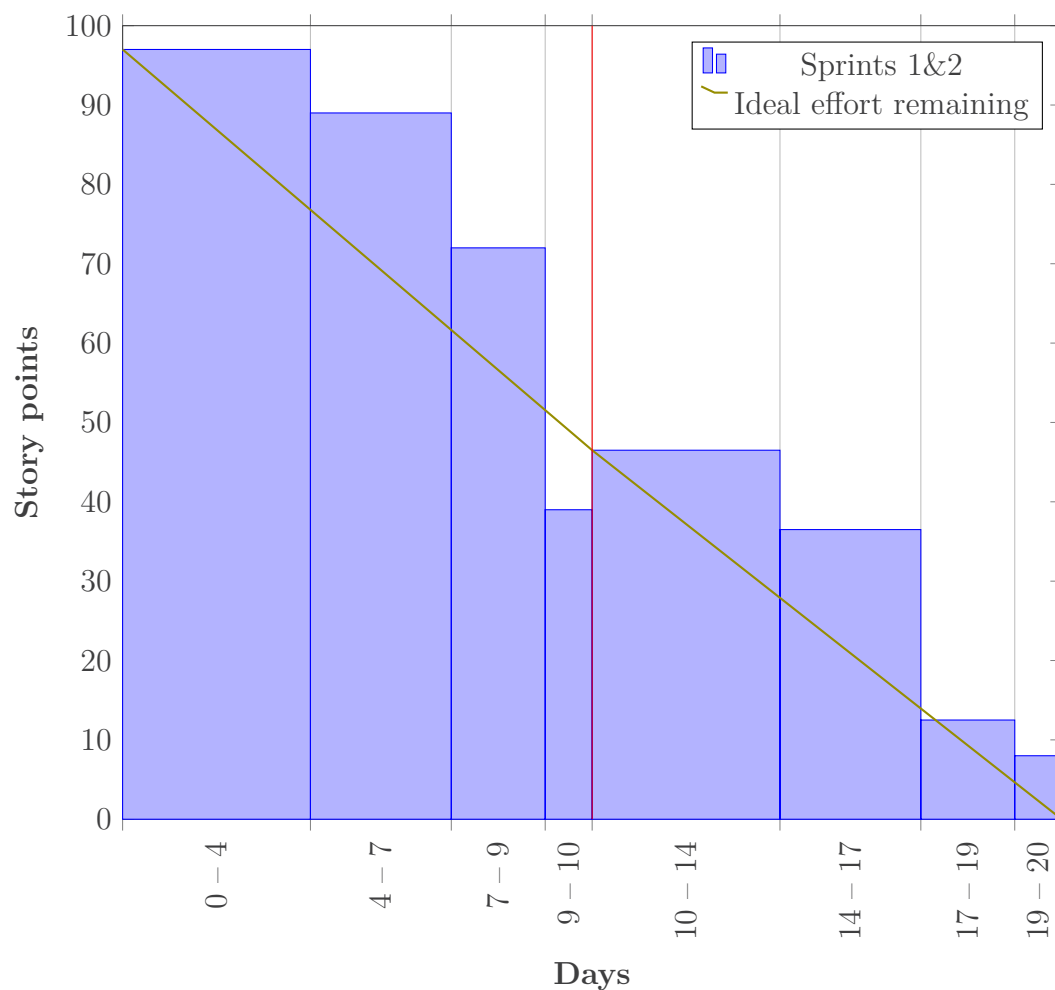


Figure 5.1: Burndown chart with the actual and estimated remaining effort over the course of Sprint 1 and Sprint 2. The red vertical line in the middle signifies the end of Sprint 1 and beginning of Sprint 2.

The Burndown chart in Figure 5.1 represents the work completed in the span of each of the sprints, with Sprint 1 running in the first ten days and Sprint 2 in the following ten. Since measurements of completed work were taken every three days, this is also represented in the chart. In Sprint 1, there were 50,5 estimated story points required to implement the command line user stories. However, 58 points were actually needed. Sprint 2 was originally planned to have 46.6 story points. Nevertheless, the eight points associated with the sessions implementation still remained at the end of Sprint 2, as discussed previously.

From the Burndown chart, we can observe that in Sprint 1 the majority of story points was completed in days 7-9. This suggests that the effort-estimation was not particularly good and extra work had to be done in those days in order to finish the sprint on time. In Sprint 2, the actual remaining effort follows the trend of the curve representing the

ideal effort remaining, which suggests that tasks were better estimated and no overtime was needed.

## 6 | Testing

### 6.1 The Testing and Quality Assurance Process

The initial test and quality assurance strategy for the project was based on that of *Extreme Programming (XP)*. The workflow can be seen in Figure 6.1 below.

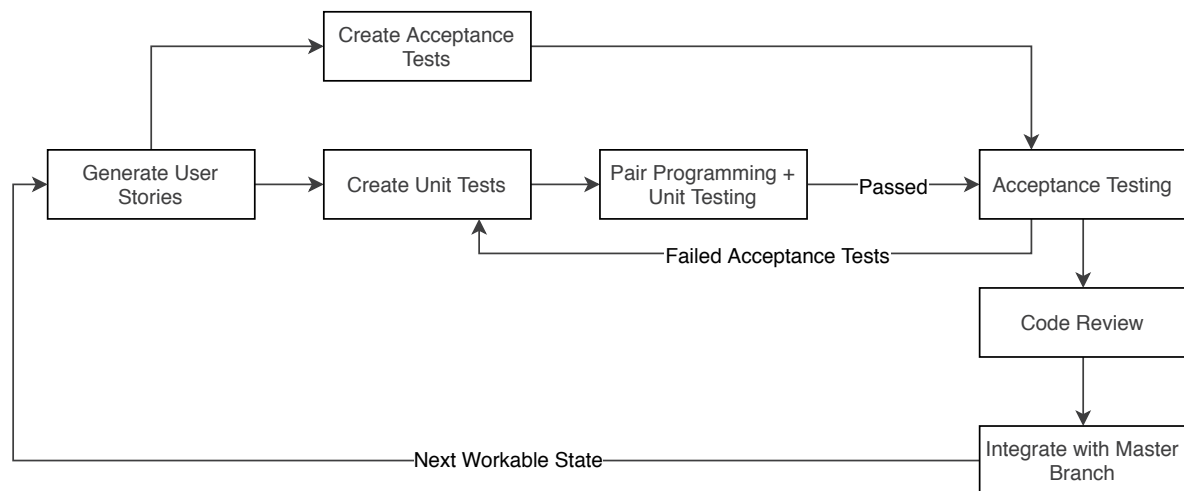


Figure 6.1: Test process. Diagram adapted from *XP Test and Analysis Strategies* [20]

On the creation of each user story, corresponding acceptance tests were documented on the card (See Chapter 2, *User Stories*). It was decided that the acceptance tests which dealt with user expectations, namely interactions with the interface(s) and the general program flow, would be asserted by manually inspecting the code during the program's execution. Acceptance tests referring to logical transitions of state were marked as potential candidates for automated testing at a later stage, as this would accelerate the development process.

In addition to the acceptance tests, it was initially decided that sufficient test coverage of a class's public methods would be a prerequisite for pushing to the master branch. This would ensure the build's integrity and avoid bugs being introduced into other branches. The *Jacoco code coverage tool* [14] was added to the Maven test and build process to allow developers to monitor areas for improvement in this respect. These tests, their accompanying resources, and the final Jacoco report can be found in the project's source code.

### 6.2 Testing Conventions

A number of conventions were established for the JUnit5 unit and integration test classes to ensure that tests were easily readable and maintainable. The `@DisplayName` annotation was used to create self-documenting, human readable test descriptions in the test reports. Additionally, `@Nested` classes were used to segregate methods requiring more than one test case. Finally, any shared variables that were used to create the appropriate testing

context would be declared at the beginning of classes, using clear capitalised naming and set as `'final'` to avoid being mutated within the tests. An example of these conventions in use is given in Listing D.1(Appendix D). No conventions were established for manual testing.

## 6.3 Implementation and Process Adjustments

### Development Bottlenecks

It became apparent during the first sprint that strict adherence to coverage was preventing functioning code being merged into master. This was because some members of the team were unfamiliar with automated testing and JUnit, resulting in delays in sharing their classes while they produced the tests. This created bottlenecks when the classes in question depended on others. It was ultimately decided that the *Github peer review process* was sufficient to ensure code quality and the coverage criteria for pushing to the master branch was relaxed. However, the team recognised the benefit of sufficient test coverage, thus a compromise was reached where a user story would not be marked as 'done' until sufficient tests existed for it.

Mocking test class dependencies was also proposed as a solution to the problem of bottlenecks. Listing D.2(Appendix D) demonstrates how the database team used the Mockito library to test their initial implementation while the `Player` class was still in development. This was found to be an effective way of creating controlled test contexts.

### Interclass Testing and Integration

Other questions that arose during the process were how to go about writing interclass tests and to what extent it would be possible to isolate tests from their dependent classes for intraclass testing. It was decided that writing mocks for every object which came from a dependent class would be too time consuming, and that a bottom-up approach to testing would need to be adopted. For example, when the required tests for `Attribute` were complete and passing, those for `Card` could be completed, and so on.

As a result of this decision, it was decided that the principle MVC classes (`TopTrumpsView`, `CliController`, `GameModel`) could be sufficiently tested manually to determine their correctness. This was because their dependencies' functionality could be assumed from prior testing and their functionality would be directly observed by the user. This was proven to be the case when a bug in `GameModel` resulted in a crash during player elimination. The removal of a `Player` from the `PlayersInGame` pool resulted in a `NullPointerException` because the loop was not being re-indexed appropriately. Because of the stage in the game it occurred, establishing its point of origin and providing a hotfix was relatively straightforward. The bug and its fix can be seen in Figures D.1 and D.2(Appendix D). No other major bugs were found when these classes were integrated, which was a testament to the overall testing and quality control processes' effectiveness.

Finally, it is worth noting that the team agreed that creating automated tests for the MVC classes was still desirable for future regression tests. However, it would be lower priority due to the amount of work that was required and the relatively diminishing returns.



## 6.4 Evaluation and Suggestions for Process Improvement

### TDD and Test Coverage

During the final reflections, it was acknowledged that a *Test Driven Development (TDD)* process which required sufficient coverage did result in far fewer bugs than adding tests after the fact. The team would no doubt enforce this practice were the project to continue, given that we are now more familiar with testing and JUnit at the end of the process. Enforcement could be achieved by integrating Jacoco with Codacy and blocking pull requests which fell under a certain coverage percentage using *Codacy's Github status checks* [21, 22].

### Mocks

The creation of more controlled tests using mock classes would result in an ability to differentiate between errors being caused by integration of classes and those caused by intraclass functionality. Going forward, establishing a convention for augmenting test cases with Mockito may be an avenue for more informative testing.

### Automated Acceptance Tests

Some automated testing of the aforementioned MVC classes was eventually implemented at the end of sprint two in the `GameModel` class. Listing D.3 (Appendix D) demonstrates an automated acceptance test for `User Story X`, specifically dealing with the state of the `Communal Pile` after a draw. The complexity of this test case vindicates the decision to test these cases manually and only implement this kind of test as a low priority refactor. A future consideration would be implementing other acceptance tests in a similar fashion.






## 7 | Assumptions

Table 7.1: *Assumptions.*

ASSUMPTIONS
* TopTrumps.json and the deck file are in the same working directory.
* Online mode is running on a modern browser which can handle ES6 features, namely template strings and arrow functions.
* The user is familiar with the game rules for Top Trumps, as there is no instructions page.
* The game server will be run on the same computer as the client that will connect to it.
* The game server will be run on a University of Glasgow PC and will be able to connect to the yacata.dcs.gla.ac.uk database.
* In cases of a discrepancy between the design specification and the output examples, the design specification is the one to be followed.
* The application does not need to handle the edge case of a draw occurring between players that are down to their last card.
* The cards to be added to the back of a round winner's deck are shuffled before being added.
* The user doesn't need to be able to change the names of the players. They are standard: USER, AI1, AI2, AI3 and AI4.
* The GUI version of the game does not require a test log.
* The CLI version of the game works with 4 AI players at all times.
* If the number of cards in the initial deck cannot be equally split among the players, the remainder is put in the communal pile.
* External dependencies can be added to the Maven project.
* The folder hierarchy of the project can be changed as long as it follows good software development practices.

## 8 | Deficiencies

Table 8.1: *Identified deficiencies and proposed solutions.*

DEFICIENCIES	
Identified Deficiency	Proposed solution
 When refactoring/adding tests, it became clear that there was some coupling in the code which made testing difficult.	✓ Further refactoring and/or the use of factory classes.
 The logger outputs initial messages to <code>System.err</code> when used with Dropwizard and certain testing suites, so has not been fully isolated from other libraries that use the <code>java Logger</code> class	✓ This does not affect the user or its functionality when writing to the log file.
 The game must be run on a UNIVERSITY OF GLASGOW PC to be able to connect to <code>theyacata.dcs.gla.ac.uk</code> database and the client accessing the server must be on the same computer as the server	✓ This could be addressed by using a different database
 When the server is accessed from multiple browser tabs to play the game, all tabs correspond to the same instance of the game.	✓ Could be solved by storing multiple <code>GameModel</code> objects in the API (created when the game is started and destroyed when the game is over). Each API method (except the one that initiates a game) would also have to be modified to accept a session id parameter which would indicate which <code>GameModel</code> object to access on the server side. The game initialization API method would have to return the session id to the client when the game starts, the client would have to save it and pass it to the server whenever it communicates with it.
 There are no instructions for the game.	✓ Adding a third option in the main screen for displaying game instructions. Another possibility is to add a tutorial option, which runs a game with pop ups for instructions and goes through different possible game scenarios.

## 9 | Conclusion

The project can be considered an overall success as the game Top Trumps can be played in both the Command Line and the Web mode with no known bugs or errors. To produce a complete version of the MVP on time, the team had to leave out the session management, which is currently not implemented as part of this submission. Given more time or simply another sprint, session management would have been implemented along with a better visual representation of the online version.

As for the learning experience, the curve for this project was quite steep, with many new technologies and tools needed to fulfill the required functionality. At first, this was very challenging for the team and caused issues such as delays during the first sprint. However, with proper communication and by deploying the strengths of Scrum, the team was able to learn from the mistakes in the first sprint and deploy a better approach during the second one. Thus, the project was delivered on time.

Overall, the team is very satisfied with the project, as it allowed us to experience Agile software development and go through a full development cycle. We believe that this knowledge is essential and will allow us to be better prepared when working on future projects in industry.

# Bibliography

- [1] Java 8. (Accessed on 16 February 2020). [Online]. Available: <https://www.oracle.com/technetwork/java/javase/overview/java8-2100321.html>
- [2] PostgreSQL. (Accessed on 16 February 2020). [Online]. Available: <https://www.postgresql.org/>
- [3] Dropwizard. (Accessed on 16 February 2020). [Online]. Available: <https://www.dropwizard.io/en/stable/>
- [4] Maven. (Accessed on 16 February 2020). [Online]. Available: <https://maven.apache.org/>
- [5] Slack. (Accessed on 16 February 2020). [Online]. Available: <https://slack.com/intl/en-gb/>
- [6] Scrum Methodology. (Accessed on 8 January 2020). [Online]. Available: <http://scrummethodology.com/>
- [7] Trello. (Accessed on 16 February 2020). [Online]. Available: <https://trello.com/en-GB>
- [8] HTML. (Accessed on 16 February 2020). [Online]. Available: <https://developer.mozilla.org/en-US/docs/Web/HTML>
- [9] JavaScript. (Accessed on 16 February 2020). [Online]. Available: <https://www.w3schools.com/js/>
- [10] Bootstrap. (Accessed on 16 February 2020). [Online]. Available: <https://getbootstrap.com/>
- [11] GitHub. (Accessed on 16 February 2020). [Online]. Available: <https://github.com/>
- [12] Mochkito. (Accessed on 16 February 2020). [Online]. Available: <https://site.mockito.org/>
- [13] JUnit 5. (Accessed on 16 February 2020). [Online]. Available: <https://junit.org/junit5/docs/current/user-guide/>
- [14] Jacoco. (Accessed on 12 February 2020). [Online]. Available: <https://www.jacoco.org/>
- [15] Codacy. (Accessed on 8 January 2020). [Online]. Available: <https://github.com/marketplace/codacy>
- [16] LaTeX. (Accessed on 16 February 2020). [Online]. Available: <https://www.latex-project.org/>
- [17] MOSCOW PRIORITISATION. Agile Business Consortium. (Accessed on 8 January 2020). [Online]. Available: [https://www.agilebusiness.org/page/ProjectFramework\\_10\\_MoSCoWPrioritisation](https://www.agilebusiness.org/page/ProjectFramework_10_MoSCoWPrioritisation)
- [18] Github branches. (Accessed on 8 January 2020). [Online]. Available: <https://gist.github.com/digitaljhelms/4287848>
- [19] A directory structure for a java project auto-generated by maven. (Accessed on 8 January 2020). [Online]. Available: [https://en.wikipedia.org/wiki/Apache\\_Maven#/media/File:Maven\\_CoC.svg](https://en.wikipedia.org/wiki/Apache_Maven#/media/File:Maven_CoC.svg)

- [20] M. Pezzè and M. Young, *Software testing and analysis - process, principles and techniques*. Wiley, 2007.
- [21] Code coverage guide - using it with codacy. (Accessed on 12 February 2020). [Online]. Available: <https://blog.codacy.com/a-guide-to-code-coverage-part-2-using-it-with-codacy/>
- [22] About status checks. (Accessed on 12 February 2020). [Online]. Available: <https://help.github.com/en/github/collaborating-with-issues-and-pull-requests/about-status-checks>

## A | Diagrams

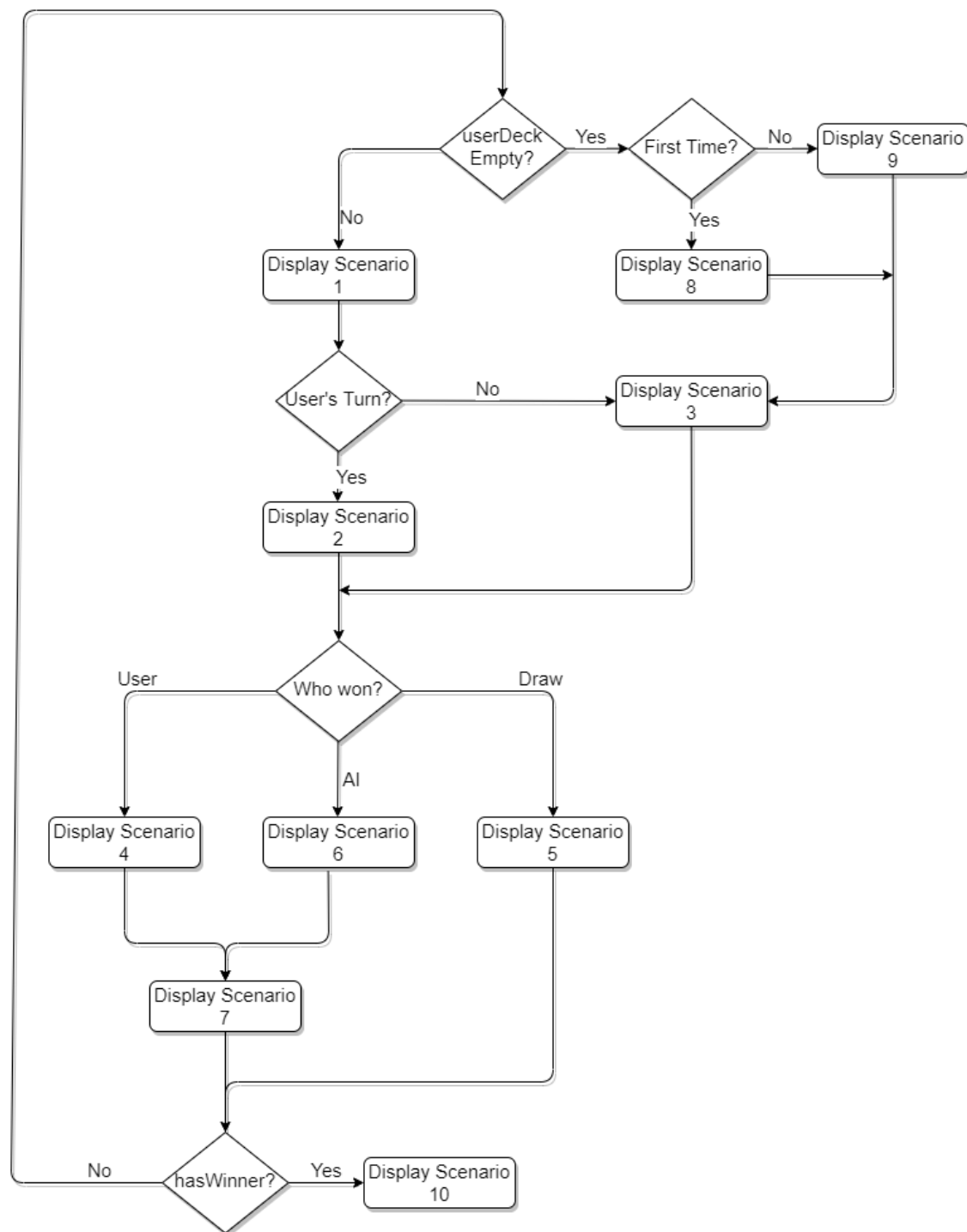


Figure A.1: TopTrumps Flowchart

## B | Functionality

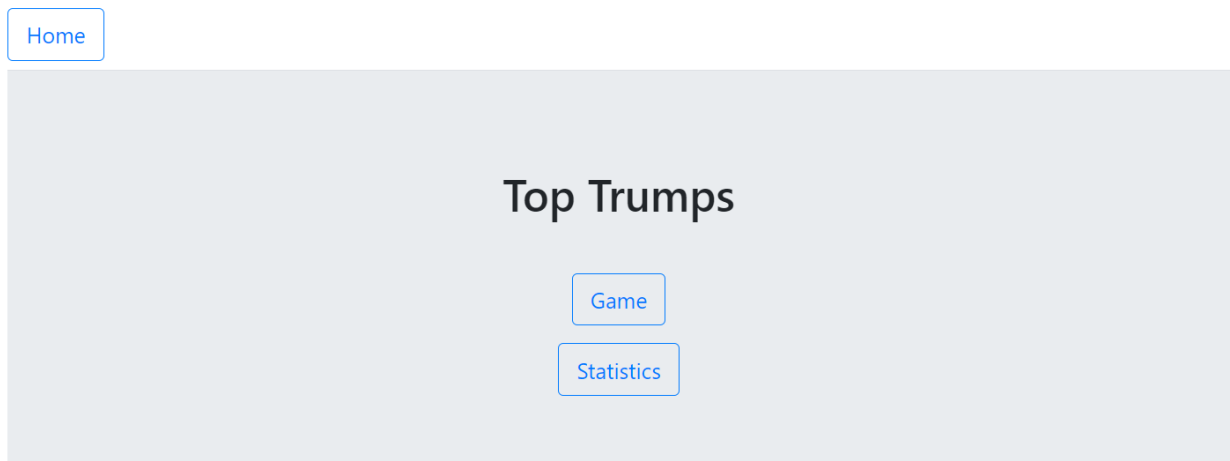


Figure B.1: Home Screen

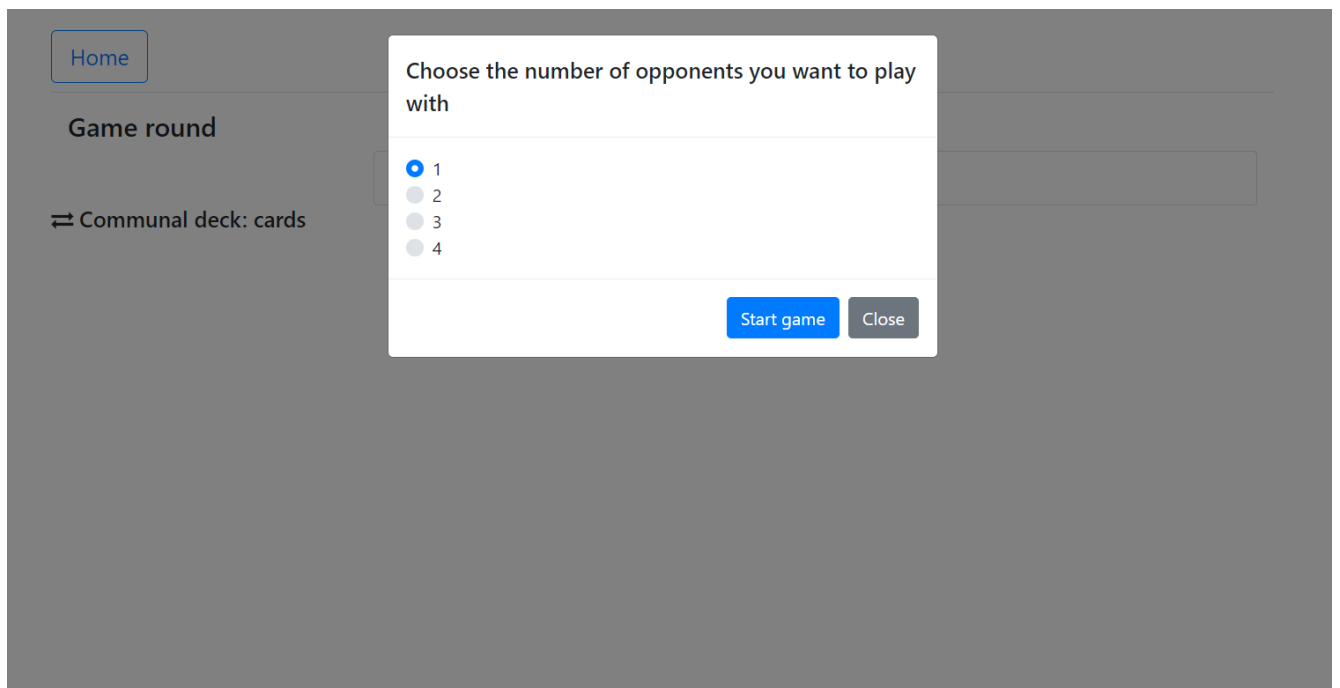


Figure B.2: Player Number Choosing



[Home](#)

**Game round 1**

Play round

AI4 is the active player

⇌ Communal deck: 0 cards

<div> <b>USER</b> Deck remaining: 7         </div> <div> <b>m50</b>            Size 1            Speed 10            Range 2            Firepower 2            Cargo 0         </div>	<div> <b>AI1</b> Deck remaining: 7         </div> <div>Top Trumps</div>	<div> <b>AI2</b> Deck remaining: 7         </div> <div>Top Trumps</div>	<div> <b>AI3</b> Deck remaining: 7         </div> <div>Top Trumps</div>	<div> <b>AI4</b> Deck remaining: 7         </div> <div>Top Trumps</div>
---	---	---	---	---

Figure B.3: Active Player for Round

[Home](#)

**Game round 1**

Next round

The round was a draw

⇌ Communal deck: 0 cards

<div> <b>USER</b> Deck remaining: 7         </div> <div> <b>m50</b>            Size 1  <b>Speed 10</b>            Range 2            Firepower 2            Cargo 0         </div>	<div> <b>AI1</b> Deck remaining: 7         </div> <div> <b>Hurricane</b>            Size 2  <b>Speed 5</b>            Range 3            Firepower 5            Cargo 0         </div>	<div> <b>AI2</b> Deck remaining: 7         </div> <div> <b>Constellation</b>            Size 4  <b>Speed 5</b>            Range 7            Firepower 3            Cargo 4         </div>	<div> <b>AI3</b> Deck remaining: 7         </div> <div> <b>Avenger</b>            Size 2  <b>Speed 5</b>            Range 4            Firepower 3            Cargo 2         </div>	<div> <b>AI4</b> Deck remaining: 7         </div> <div> <b>m50</b>            Size 1  <b>Speed 10</b>            Range 2            Firepower 2            Cargo 0         </div>
--	--	--	--	---

Figure B.4: Round Draw

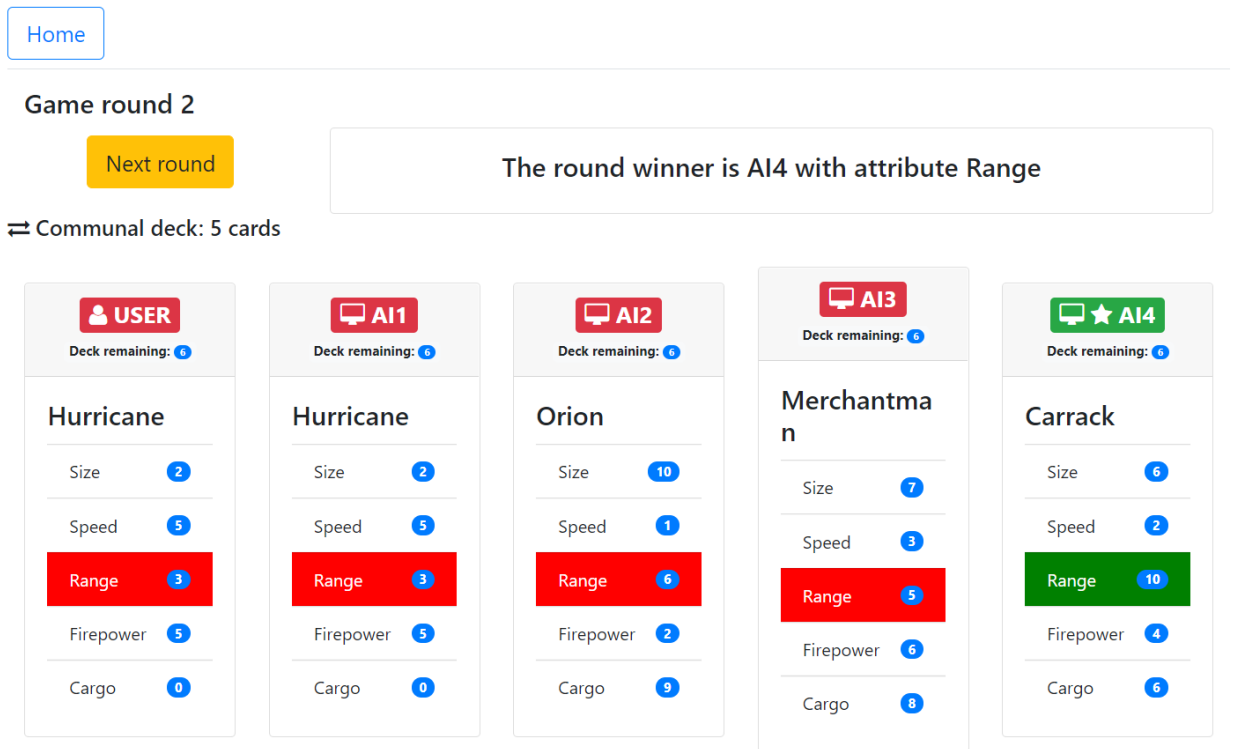


Figure B.5: AI Round Winner

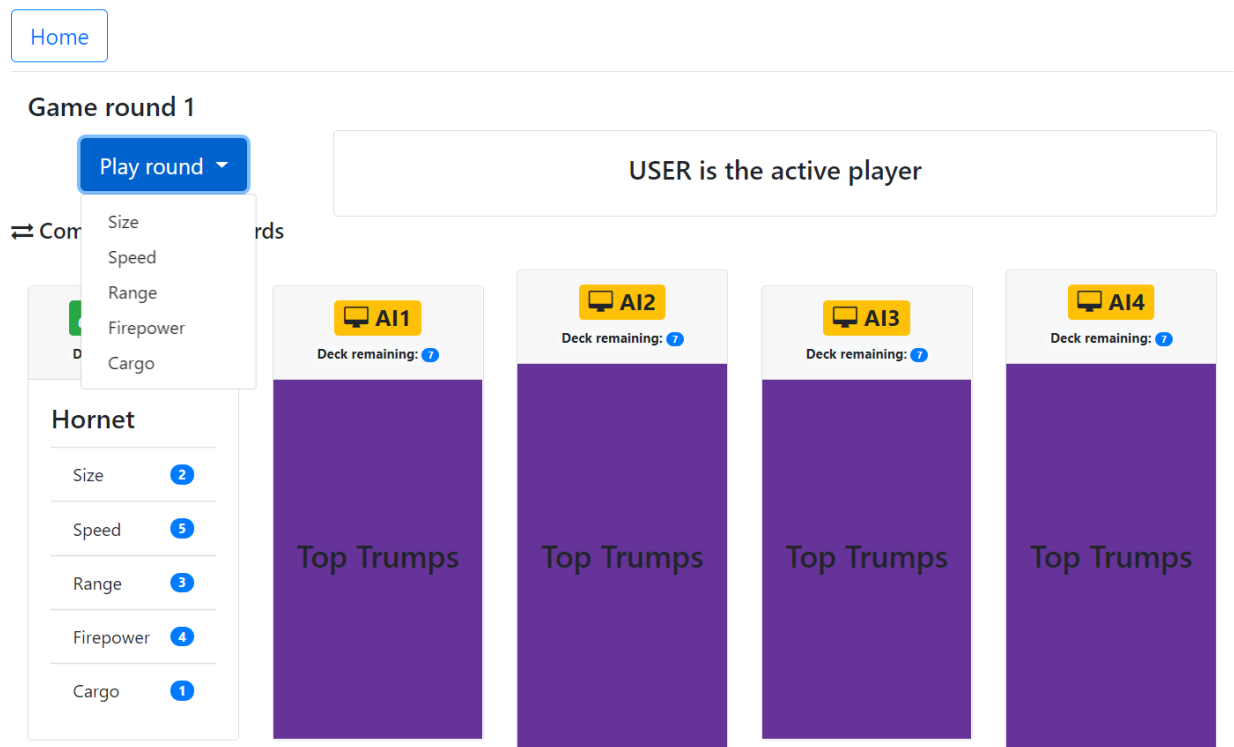


Figure B.6: User Attribute Choosing

## Game round 1

Next round

The round winner is USER with attribute Range

⇒ Communal deck: 0 cards




<div>   <b>USER</b> </div> <div>Deck remaining: 19</div>	<div>  <b>AI1</b> </div> <div>Deck remaining: 19</div>
Carrack	m50
Size 6	Size 1
Speed 2	Speed 10
Range 10	Range 2
Firepower 4	Firepower 2
Cargo 6	Cargo 0

Figure B.7: User Round Winner

Figure B.8: Auto Complete When User Loses

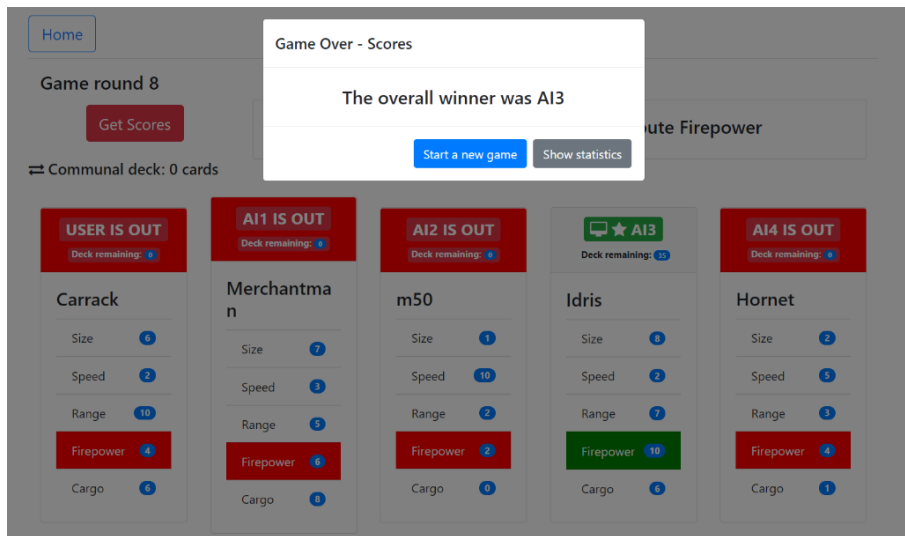


Figure B.9: AI Game Winner

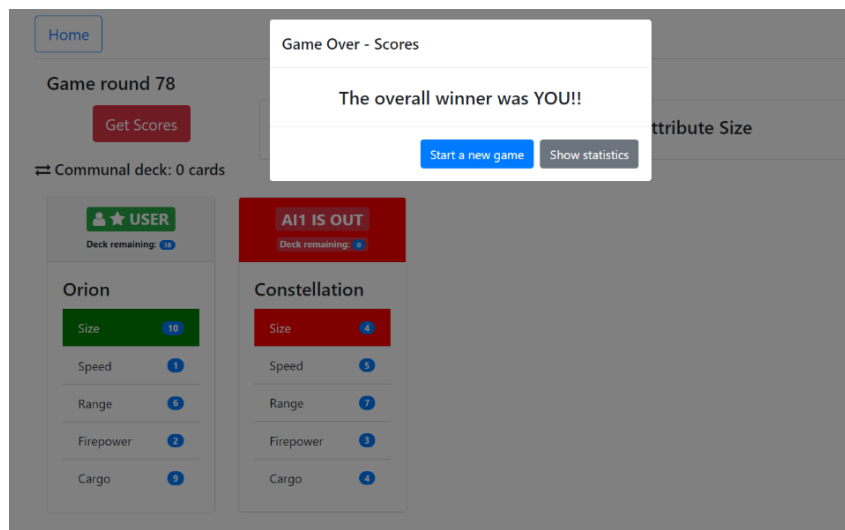


Figure B.10: User Game Winner



Figure B.11: User Game Stats

# C | Screenshots

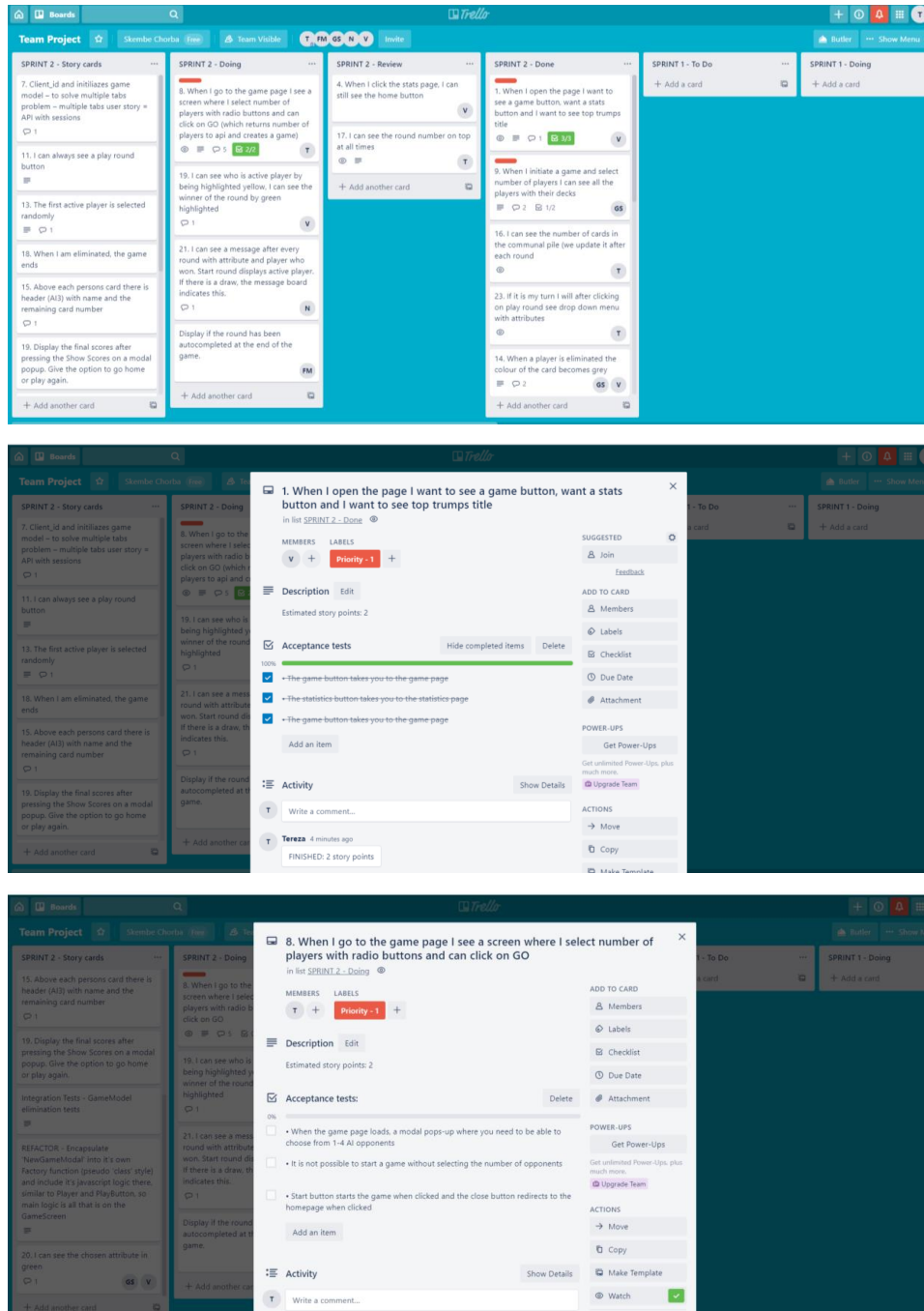


Figure C.1: Trello story-cards screenshot

## D | Tests

```
01 |  
02 | /**  
03 |  * Card tests  
04 |  */  
05 |  
06 | public class CardTest {  
07 |  
08 |     // An example of recycled test variables  
09 |     protected static final Attribute ATTRIBUTE_STRENGTH_1 =  
10 |         new Attribute("Strength", 1);  
11 |     protected static final Attribute ATTRIBUTE_STRENGTH_5 =  
12 |         new Attribute("Strength", 5);  
13 |     protected static final Attribute ATTRIBUTE_STAMINA_11 =  
14 |         new Attribute("Stamina", 11);  
15 |     protected static final Attribute ATTRIBUTE_MONEY_4 =  
16 |         new Attribute("Money", 4);  
17 |  
18 |     // ...  
19 |  
20 |     // An example of a nested class testing one class method  
21 |     @Nested  
22 |     @DisplayName("getAttributes()")  
23 |     class GetAttributes {  
24 |  
25 |         /*  
26 |          * Validation Tests  
27 |          */  
28 |  
29 |         // An example of an overridden test method name  
30 |         @DisplayName("Returns an attribute list")  
31 |         @Test  
32 |         public void getsAttributeList() {  
33 |             Card testCard = new Card("Test");  
34 |  
35 |             testCard.add(ATTRIBUTE_STRENGTH_1);  
36 |             testCard.add(ATTRIBUTE_STAMINA_11);  
37 |             testCard.add(ATTRIBUTE_MONEY_4);  
38 |  
39 |             ArrayList<Attribute> expectedAttributeList =  
40 |                 new ArrayList<Attribute>();  
41 |             expectedAttributeList.add(ATTRIBUTE_STRENGTH_1);  
42 |             expectedAttributeList.add(ATTRIBUTE_STAMINA_11);  
43 |             expectedAttributeList.add(ATTRIBUTE_MONEY_4);  
44 |  
45 |             assertEquals(expectedAttributeList,  
46 |                 testCard.getAttributes());  
47 |         }  
48 |  
49 |         // ...
```

Listing D.1: Naming conventions

```

01 |
02 | /**
03 |  * Database tests
04 |  */
05 | class DatabaseTest {
06 |
07 |     //...
08 |
09 |     // Mockito usage is demonstrated below.
10 |
11 |     // Mock objects are created using the mock() method.
12 |     // The return behaviour is declared for
13 |     // the invocation of a particular method.
14 |
15 |     // Set up mock players for test.
16 |     user = mock(Player.class);
17 |     ai1 = mock(Player.class);
18 |     ai2 = mock(Player.class);
19 |     ai3 = mock(Player.class);
20 |     ai4 = mock(Player.class);
21 |
22 |     // Set up player array for test.
23 |     players = new Player[] {user, ai1, ai2, ai3, ai4};
24 |
25 |     // Set up names of players.
26 |     when(user.toString()).thenReturn("USER");
27 |     when(ai1.toString()).thenReturn("AI1");
28 |     when(ai2.toString()).thenReturn("AI2");
29 |     when(ai3.toString()).thenReturn("AI3");
30 |     when(ai4.toString()).thenReturn("AI4");
31 |
32 |     // Set the roundsWon for the players in the first game for the test
33 |     .
34 |     when(user.getRoundsWon()).thenReturn(2);
35 |     when(ai1.getRoundsWon()).thenReturn(8);
36 |     when(ai2.getRoundsWon()).thenReturn(3);
37 |     when(ai3.getRoundsWon()).thenReturn(3);
38 |     when(ai4.getRoundsWon()).thenReturn(4);
39 |
40 |     //...
41 |
42 |     // Upload the first game to the database.
43 |     database.uploadGameStats(2, 20, "AI1", players);
44 |
45 |     //...

```

Listing D.2: Mocking the Player class in DatabaseTest

```

01 | @DisplayName("Communal Pile empties after a draw.")
02 | @Test
03 | void communalPileResetAfterWin() {
04 |     int numAIPlayers = 4;
05 |     GameModel model = new GameModel(CARD_DECK_40, numAIPlayers);
06 |
07 |     // PLAY THE GAME OVER AND OVER UNTIL A DRAW OCCURS
08 |     Player activePlayer;
09 |     Attribute chosenAttribute;
10 |
11 |     // Continue playing until there is a draw
12 |     while (model.getDraws() != 1) {
13 |         activePlayer = model.getActivePlayer();
14 |         chosenAttribute = activePlayer.peekCard().getAttribute(0);
15 |         model.playRoundWithAttribute(chosenAttribute);
16 |
17 |         if (model.checkForWinner() != null) {
18 |             model.reset();
19 |         }
20 |     }
21 |
22 |     // After a draw, the communal pile should have cards.
23 |     assertTrue(model.getCommunalPileSize() > 0);
24 |
25 |     // Play until there is a win.
26 |     do {
27 |         activePlayer = model.getActivePlayer();
28 |         chosenAttribute = activePlayer.peekCard().getAttribute(0);
29 |         model.playRoundWithAttribute(chosenAttribute);
30 |     } while (model.getDraws() != 1);
31 |
32 |     // THEN ASSERT THAT THE COMMUNAL PILE SIZE IS 0
33 |     assertEquals(0, model.getCommunalPileSize());
34 | }

```

Listing D.3: An automated acceptance test for User Story X



```

USER has 2 cards remaining!
---
AI2 is the active player!
AI2 drew 'Vanguard':
  > Size: 3
  > Speed: 4
  > Range: 5
  > Firepower: 5
  > Cargo: 2
---
The category chosen is: Range
Round 8 was a draw.
The communal pile now has 5 cards
USER has been eliminated!
---
AI3 has been eliminated!
---
Round 8
Players have drawn their cards
---
AI2 is the active player!
AI2 drew 'Orion':
  > Size: 10
  > Speed: 1
  > Range: 6
  > Firepower: 2
  > Cargo: 9
---
The category chosen is: Size
Exception in thread "main" java.lang.NullPointerException
    at model.GameModel.playRoundWithAttribute(GameModel.java:111)
    at commandline.controller.CliController.playRound(CliController.java:89)
    at commandline.controller.CliController.run(CliController.java:42)
    at commandline.TopTrumpsCLIApplication.main(TopTrumpsCLIApplication.java:34)
bash-3.2$

```

Figure D.1: Integration bug caused by GameModel [20]

164 - /**		164 /**
165 - * Removes player from players in game		165 * Checks whether human player is still in game
166 - */		166 */
167 - public void eliminatePlayer(Player eliminated) {		
168 - playersInGame.remove(eliminated);		
169 - }		
170 -		
171 /**		171 /**
172 * Checks whether human player is still in game		172 * Checks whether human player is still in game
173 */		173 */
174 @@ -195,12 +188,19 @@ public Player checkForWinner() {		
175		
195 - /**	188 + /**	
196 - public ArrayList<Player> checkToEliminate() {	189 + public ArrayList<Player> checkToEliminate() {	
197 - ArrayList<Player> eliminated = new ArrayList<Player>();	190 + ArrayList<Player> eliminated = new ArrayList<Player>();	
198 - for (int i = 0; i < playersInGame.size(); i++) {	191 +	
199 - if (playersInGame.get(i).peekCard() == null) {	192 + // Check which players are to be eliminated	
200 - eliminated.add(playersInGame.get(i));	193 + for (Player player : playersInGame) {	
201 - eliminatePlayer(playersInGame.get(i));	194 + if (player.peekCard() == null) {	
	195 + eliminated.add(player);	
202 - }	196 + }	
203 - }	197 + }	
	198 +	
	199 + // Eliminate players from model	
	200 + for (Player eliminatedPlayer : eliminated) {	
	201 + playersInGame.remove(eliminatedPlayer);	
	202 + }	
	203 +	
204 - return eliminated;	204 + return eliminated;	
205 - }	205 + }	
206	206	

Figure D.2: Integration bug origin in GameModel [20]