# Rock Paper Scissors – eBay coding puzzle

James Siddle, 16th September 2012

This is a living document describing the requirements, design, implementation, and testing of my Rock Paper Scissors (RPS) coding challenge. As I work through the coding exercise I’ll add to this document. Hopefully this will provide a helpful insight into the final product, and the rationale behind key decisions.

## Planning

After reviewing the user stories, discussing the game with a few friends, and researching computer-computer RPS games, I’ve settled on the following plan of action:

1. **Interpret / elaborate requirements** - the plan is to start by interpreting the requirements, and describing how I expect the final product to look – ideally I’d discuss the requirements with a customer at this point.
2. **Outline conceptual domain model** - next, I’ll apply a DDD approach and create a conceptual domain model to underpin further development. This should ensure I’ve captured the key principles and concepts in the domain, and can act as a communication vehicle.
3. **Rough up-front design** - I’ll use a little up-front design to identify key components of the core implementation, then develop those using a strictly Test Driven approach.
4. **Develop core implementation** using TDD
5. **Get feedback, revise, add features** - once I have a working game, I’m get feedback by putting the game in front of my housemate, Ali. I’ll then update the implementation based on his feedback (if necessary) then work on extra features.
6. **Ensure code quality using Emma and FindBugs** - finally, I’ll generate some quality metrics using Emma and FindBugs, showing unit test coverage and potential bugs.

*Notes:*

I considered using BDD, but it’s not something I’m very familiar with so decided to stick with just TDD to keep things simple.

## 1. Interpret Requirements

I tend to lean towards back end development rather than UI, so I’m planning to create a simple **console based game**, and focus on an interesting game experience rather than a nice looking front end.

I’ll start with the **core game concept** (“Rock, Paper, Scissors”), but allow **extensions** including the “… Lizard, Spock” variant, and other examples such as “Ranger, Poacher, Bear” played in the US, as well as higher-cardinality variants. I’m treating extensibility as a core architectural constraint so the customer can expect it from day 1.



Picture courtesy Viola Burke’s [Flickr stream](http://www.flickr.com/photos/violaburke/5269137284/).

The player will be able to **challenge the computer**, or watch a **computer-computer match**. I’ll provide a couple of **different computer player** types, so the human player can pit different algorithms against each other.

I’m also planning to use a little

**\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ .\_\_\_.\_\_\_ \_\_\_\_\_ \_\_**

**/ \_ \ / \_\_\_\_\_/\\_ \_\_\_ \| | | / \_ \\_\_\_\_\_\_\_\_/ |\_**

**/ /\_\ \ \\_\_\_\_\_ \ / \ \/| | | / /\_\ \\_ \_\_ \ \_\_\**

**/ | \/ \\ \\_\_\_| | | / | \ | \/| |**

**\\_\_\_\_|\_\_ /\_\_\_\_\_\_\_ / \\_\_\_\_\_\_ /\_\_\_|\_\_\_| \\_\_\_\_|\_\_ /\_\_| |\_\_|**

**\/ \/ \/ \/**

to liven things up a bit.

## 2. Conceptual Domain Model

The following diagram shows the key concepts and their relationships from the product domain. I’ve used a UML class diagram notation (created in MS Word for simplicity) to capture the concepts. This OO approach can be used to describe concepts as a precursor to coding, or more implementation focussed modelling.

The model includes: concepts, attributes, associations, cardinality, and role descriptions. Directionality and ownership / containment are omitted:

2

3+

applies

2

plays

participate in

**Token**

defined by

**Rule**

**Game**

Num rounds

Results

**Player**

1+

The model has four elements: **Game**, **Rule**, **Player**, and **Token**.

A game has two players, who participate in the game. A game has three or more Rules, which are each defined by two tokens. Tokens represent game elements such as Rock, Paper, etc. A game has a number of rounds during which a player can play tokens; the rules define which token wins. The game also has results, including who won each round and who is the overall winner.

## 3. Rough Up-front Design

Following the core Agile principle of developing iteratively and in small increments, I’ve done a little up-front design work. This should be sufficient for the one or two planned iterations.

The core implementation will consists of the following components:

1. **Rule Interpreter** – responsible for parsing a file of RPS rules, provides an interface for checking player moves.
2. **Game Engine** – controls game flow, requests moves from players, checks and records outcomes of each round and the overall game
3. **Players** – must provide moves to the game engine on request, will be notified of outcomes each round
4. **Console** – allows the player to start games, provide moves, and observe game outcomes

I’ve also identified a few software design patterns that may prove useful during implementation:

*Model View Controller* – the MVC pattern can help inform the structure of the relationship between the Game Engine and the console input and display elements.

*Observer* – often applied as part of MVC, it may be worth using Observer to decouple the game engine from display components, to allow new UIs to be introduced easily.

A *Strategy*-like approach can be used to implement players, so that different types of player can be easily configured. This will be immediately useful for handling human and computer types, and would also allow the addition of remote players at a later point.

*Dependency Injection* – will be used to support Test Driven Development.

## 4. Develop Implementation

I’ll be using the following tools to develop the game:

* **Eclipse** (Indigo)
* **Java 6[[1]](#footnote--1)**
* **JUnit** – for unit testing
* **Mockito** – for mocking
* **GitHub** – for source control

The source code can be found on GitHub, and should also be included with this document: See the in-source JavaDoc and comments for code level documentation.

Some development notes:

* I’m creating an explicit Token class; could use String but this seems cleaner.
* Only performing rudimentary checking of edge cases

To run the game:

1. I considered using Java 7, but the Mac OS X version has only just been released and I know and trust the Java 6 JVM currently installed. [↑](#footnote-ref--1)