

# **System design document for Project Dragon (SDD)**

## [1 Introduction](#)

### [1.1 Design goals](#)

### [1.2 Definitions, acronyms and abbreviations](#)

## [2 System design](#)

### [2.1 Overview](#)

#### [2.1.1 Design against interfaces](#)

#### [2.1.2 Law of Demeter](#)

#### [2.1.3 Security](#)

### [2.2 Software decomposition](#)

#### [2.2.1 General](#)

#### [2.2.2 Decomposition into subsystems](#)

#### [2.2.3 Layering](#)

#### [2.2.4 Dependency analysis](#)

### [2.4 Persistent data management](#)

## [3 References](#)

Version: 1.2

Date 2012-05-22

Author anrobin, lisast

This version overrides all previous versions.

# 1 Introduction

## 1.1 Design goals

The design must have loose and few connections between classes to reduce dependencies. This is important to allow for easy expansion with new functionality, new card games like Omaha poker for example, and to be able to easily adapt the application to a client/server architecture. The models and controllers should be testable. See RAD for usability.

## 1.2 Definitions, acronyms and abbreviations

Technical vocabulary:

- *GUI* - Graphical User Interface
- *Java* - platform independent programming language
- *JRE* - the Java Run time Environment.
- *Host* - a computer where the game server will run
- *Client* - the application that is used on each computer to manage ones account and connect to the host.
- *MVC* - a design pattern where all classes belongs to one of the three categories model, view or control. Model classes contain all the data and relevant methods that describes the different states of the application, whereas the view classes contains all the GUI components. The controller classes handles the applications actions - controls the program - by handling the communication between the different classes.

All definitions regarding poker follow standard poker vocabulary:

- *Poker chips (or simply "chips")* - the currency of the game
- *Pot* - every poker chips that players have bet during a round (see below).
- *Bet* - an amount of poker chips that is put on stake by a player to try to win the pot
- *Raise* - a bet with more chips than the current highest bet (during a betting round, see below). The raise becomes the new highest bid.
- *Call* - a bet where a player inputs the exact same amount as the highest bet.
- *Check* - no bet (raise) when the highest bet is the same as the current players last bet.
- *Fold* - the player throws away his cards and forfeits the betting round
- *All in* - a player bets (either by raise or call) all his chips
- *Round* - one round ranging from the blinds to the showdown or when all but one player has folded. Ends with the distribution of chips
- *Betting round* - a short round that starts with the first player choosing to check, raise, call or fold. It ends when all players have called the highest bet or folded.
- *Buy in* - a player that inputs more "real" money and buys himself more chips because he has already lost all his chips but wants to continue the game. Buy ins are not always allowed.
- *Dealer button* - a button (or big "chip") that circulates clockwise among the players at a poker table. The player with the dealer button should be the dealer, although that assignment is usually handled by a dedicated dealer that does not participate in the

competition. Who has the blinds (see below) and who the dealer distributes the first card to is determined with the dealer button as starting-point.

- *Blind, small blind and big blind* - an amount of chips that the first player (small blind) and second player (big blind) to the left of the dealer button has to bet at the beginning of a round. This is done to force players to bet something at certain intervals so as to eliminate the option for a player to simply check his way through an entire game.
- *Hand* - The two cards that each player is given during the start of each round.
- *Showdown* - when all players have checked, called or folded at the end of the last betting round every remaining player shows their hand to determine the winner - who is then given the pot.
- *Flop* - the three cards that the dealer puts on the table after the first betting round.
- *Turn* - The fourth card that the dealer puts on the table after the second betting round.
- *River* - The fifth and final card that the dealer puts on the table after the third betting round

## 2 System design

### 2.1 Overview

The program will follow the MVC design model but with a few large control classes divided into two areas of responsibility - server- and client controllers.

#### 2.1.1 Design against interfaces

To allow for easy expansion we have designed against interfaces where needed. For example we have an interface for Hand (iHand), thus it's possible to expand with other poker-games. We also have an interface for a Dealer (iDealer) for the same sake.

The interface iPlayer allows us to do the same actions on Player and User where needed (especially needed in Table). iCard is an interface for different Cards. We have this interface because a client should only have access to its own cards and not the others.

#### 2.1.2 Law of Demeter

When designing classes we take count of the "Law of Demeter". This means that a class should know as little as possible about other classes internal structure. However this was not applicable in every case, for instance the controller. The controller is allowed to know everything about the model.

#### 2.1.3 Security

We do not let a player see the other players hands until showdown. Otherwise we do not consider security-problems.

## 2.2 Software decomposition

### 2.2.1 General

`model` is the top level package for all model related classes. `client.model` is the top level package for the client model, which holds just one class: `Table`. This because this is the only thing that differ the server model from the client model.

*`common.model.player`*: Classes needed to represent a player (such as `iPlayer`, `Account` et.c)

*`common.model.player.hand`*: Classes needed to represent a hand with Cards (`iHand`, `HandEvaluator` et.c), except `Card` that has its own package.

*`common.model.card`*: `Deck`, `Cards` and `Comparator` for `Cards`.

*`common.utilities`*: Exceptions and constants

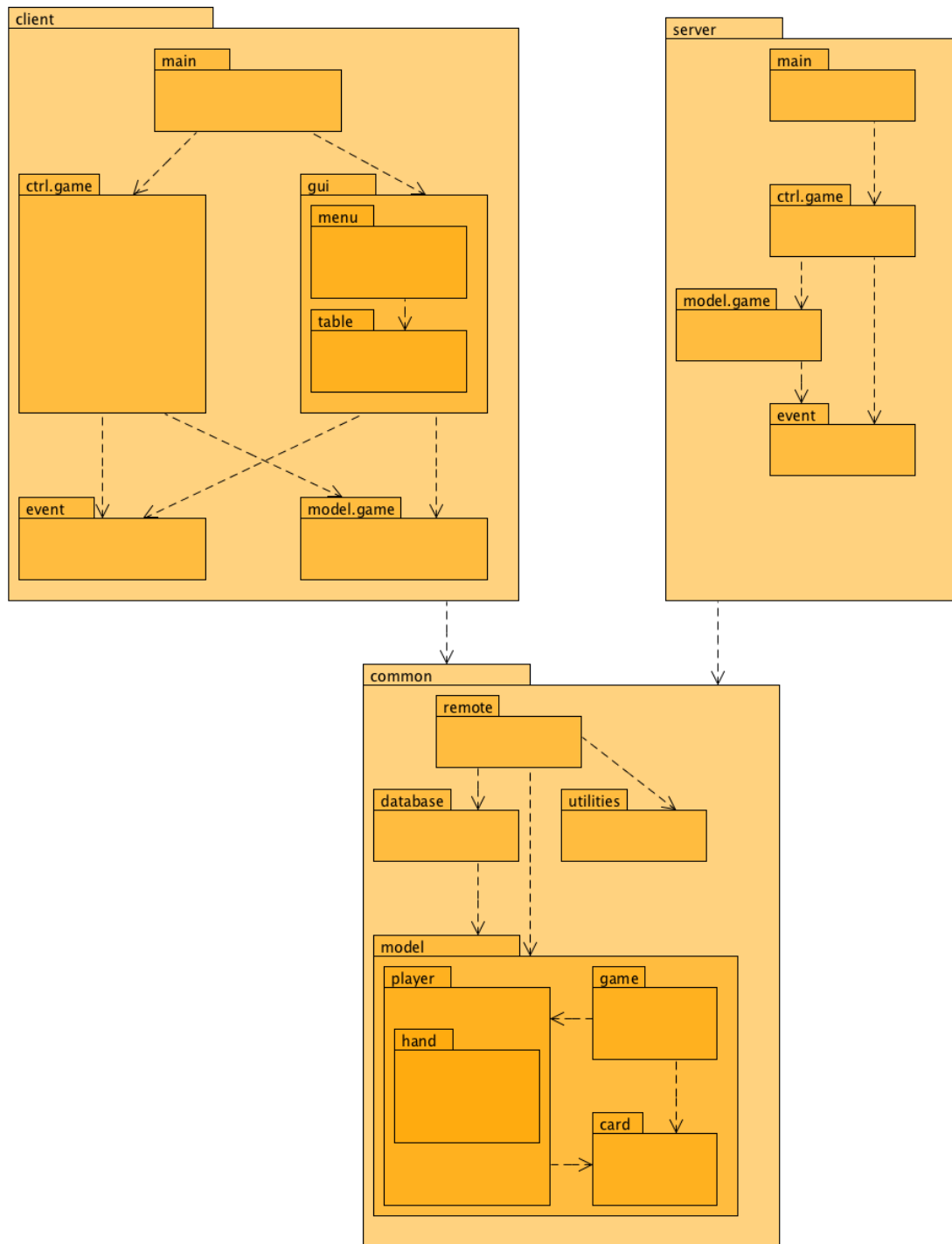
*`common.database`*: Driver for the database and interfaces needed to connect to the database.

*`server.event`*: `Eventbus`, `EventHandler` and `Event` needed for the Server.

*`server.ctrl.game`*: Controller for the server.

*`client.event`*: `Eventbus` et.c needed for the client.

*`client.ctrl.game`*: Controller for the client.



*Higher Level design.*

## 2.2.2 Decomposition into subsystems

We are using a database to save information about accounts and games. An account has username as its key. In other words the username defines all the other values. A game is recognized by its gameId. The game-table does at the present moment just save information about when the game was played, but we think that this is expandable for later purpose.

The table PlayedGames saves information about placements in games. This table has gameId and player as its keys.


Column	Type	Not Null	Default	Constraints	Actions			Comment
username	character varying(50)	NOT NULL			<a href="#">Browse</a>	<a href="#">Alter</a>	<a href="#">Drop</a>	
firstname	character varying(50)				<a href="#">Browse</a>	<a href="#">Alter</a>	<a href="#">Drop</a>	
lastname	character varying(50)				<a href="#">Browse</a>	<a href="#">Alter</a>	<a href="#">Drop</a>	
password	character varying(50)				<a href="#">Browse</a>	<a href="#">Alter</a>	<a href="#">Drop</a>	
balance	integer				<a href="#">Browse</a>	<a href="#">Alter</a>	<a href="#">Drop</a>	

Table: Accounts


Column	Type	Not Null	Default	Constraints	Actions			Comment
gameid	character varying(50)	NOT NULL			<a href="#">Browse</a>	<a href="#">Alter</a>	<a href="#">Drop</a>	
date	character varying(50)				<a href="#">Browse</a>	<a href="#">Alter</a>	<a href="#">Drop</a>	

Table: Games



Column	Type	Not Null	Default	Constraints	Actions			Comment
gameid	character varying(50)	NOT NULL			<a href="#">Browse</a>	<a href="#">Alter</a>	<a href="#">Drop</a>	
player	character varying(50)	NOT NULL			<a href="#">Browse</a>	<a href="#">Alter</a>	<a href="#">Drop</a>	
placement	integer				<a href="#">Browse</a>	<a href="#">Alter</a>	<a href="#">Drop</a>	

Table: PlayedGames

We also have a view: WonGames that contains information about players and how many games they have won. This data is calculated with help from the table PlayedGames.

## 2.2.3 Layering

The layering is as indicated in Figure (missing) . Higher layers are at the top of the gure.

## 2.2.4 Dependency analysis

Dependencies are as shown in Figure (missing). There are no circular dependencies except between core and core.spaces (no problem since this is just a administrative separation of classes representing spaces).

## 2.4 Persistent data management

See 2.2

### 3 References

Law of Demeter.

<http://www.cse.chalmers.se/edu/year/2012/course/TDA550/Lectures/F7-OH.pdf>

MVC, see <http://en.wikipedia.org/wiki/Model-view-controller>

Texas Hold'em Hand Evaluator. [HandEvaluator, HandValue, HandValueType]

<http://code.google.com/p/texasholdem-java/source/browse/>

[#svn%2Ftrunk%2Ftexasholdem%2Fsrc%2Fmain%2Fjava%2Forg%2Fozsoft%2Ftexasholdem%253Fstate%253Dclosed](http://code.google.com/p/texasholdem-java/source/browse/#svn%2Ftrunk%2Ftexasholdem%2Fsrc%2Fmain%2Fjava%2Forg%2Fozsoft%2Ftexasholdem%253Fstate%253Dclosed)



