System General Description

The system is a multiplayer Texas Hold'em[1] game center. Players connect to the game center via a game client and may join multiple active poker games as either players or spectators.

The **game center** includes multiple **game instances**, each game instance represents an active texas hold'em game. Each game instance is played in "Hotseat" style, meaning **players** take turns and cannot perform actions simultaneously. Additionally, every game instance has its own set of **game preferences** which holds information about the game, such as game type, buy-in policy (the cost of joining the game) etc. Game instances are saved in **game log**s in order to support spectating a game and viewing of **game replays** even after the games have finished.

The game center manages **leagues**, in which players compete against players of similar skill and experience. New players are assigned to the lowest league and are able to progress over time. Players can also be demoted to lower leagues due to lack of activity or losing too much.

In order to **create** or **join** a game, one needs to **register** to the system and become a **user** and to be **logged in**. Users can edit their personal information in their **user profile.** In addition, the system allows users to **find active games** to join or spectate according to different criteria.

The system also supports in game communication between the players, and can produce a variety of statistics and other information based on the different games.

Assignment 0

The purpose of this assignment is to **analyze the requirements** and **model** the main functionality of the system (the **domain layer** of the system). It contains the **initial** requirements. Note that later versions will include additional requirements, as well as modifications of existing requirements. Therefore, it is in your best interest to design a system that is robust to changes, hence with high cohesion, and low coupling. After each assignment, you will present your work to your client (your adviser).

Business and Integrity Rules

Users

- Each user is identified by a unique identifier (such as a username).

[1] See Texas Hold'em game description document

- Users maintain a collection of games in which they participate.
- Each user has a wallet, the value of which should never be allowed to drop below 0$.
- Users must be able to retrieve the active games in which they are participating, even after logging out of the system and later logging back in.

Games

- Games consist of at least two players.
- The payout at the end of a game must must be equal to the sum of money put into it by the by the players.
- Cards must be dealt from a proper deck.

Notifications

- Notifications must be restricted to the set of users to whom the notifications are relevant. The only notifications that should be published to all users are system-wide notifications (e.g. “The system will be down for maintenance for two hours on Monday”).
- Notifications should be instant for logged-in users.
- Notifications should be delayed for non-logged-in users, and displayed to them immediately upon logging in to the system.

Logging

- An action log should be maintained. All actions performed on the system should be recorded in this log.
- An error log should be maintained. All errors which occur system should be recorded in this log.

Security

- The privacy of users should be maintained at all times.
- Secret information (such as passwords) must be stored in a secure fashion, making it impossible for anyone to restore such information.
- User access should be restricted to functionality and information that is allowed to them.

Functional Requirements

Users

- Register to the system.
- Login and logout from the system.
- Edit user profile which should include an avatar (image of their choice).
- Create Texas Hold'em games.

- Join existing games.
- Spectate active game.
- Leave a game.
- Replay games that are no longer active.
- Save favourite turns from replays in order to be viewed on later occasions.
- Find all active games which the user can join.

Game Center (System)

- Store all the information from a game, such as: actions performed by all players in the game, the cards dealt at each round, round beginning and end, etc.
- List all active games which are available for spectating.
- Maintain leagues, managing which users are in which league at any given moment.

Game

- support playing a Texas Hold'em game: dealing cards, placing blind bets for players, allowing players to check (NOP), fold or bet according to the game rules, etc.

Nonfunctional Requirements

Speed (waiting time)

The system should be light and fast to react. Users should not be required to wait for actions and processes to complete. Interactive actions should react in real-time.

Data Integrity

No data should be lost or deleted, except for special cases in which users are allowed to instruct the system to discard information (e.g. deleting a profile picture).

Load

The system must support multiple concurrent connected users.

Availability and Stability

Once the system is set up and initiated, it must remain online and reactive throughout the required duration (until the system is manually shut down).

Obviously, this excludes uncontrollable external events, such as power and network outages.

Capacity

The game system must be able to support an arbitrary number of concurrent active games, each active game must be able to support an arbitrary number of players, and the system must be able to support an arbitrary number of active replays.

Mobility

The system should be simple to install and run on a variety of hardware.

Usability

User Interface

- Users will communicate with the game system using a graphical user interface. This interface must be comfortable for both experienced and inexperienced users.
- Buttons in the interface should contain only short text descriptions (no more than two words) or a descriptive icon (preferably both).
- UI elements should have tooltips attached which provide a simple explanation of each element's function and purpose.
- Text fields and text boxes should be large enough to comfortably accommodate user input, but not so large that they are unwieldy.
- Make sure to use combo-boxes where appropriate (input is constrained to a finite set of values).
- Data input by users should be validated, and input of invalid characters should be blocked (e.g. users should not be able to type letters into a phone-number field).
- For additional guidelines, read this page on Good User Interface Design Tips.

System Feedback

Users should have the most current information on the status of their active games, their user profile, and any other relevant information.

In case of an error, users should receive meaningful notification from the system, explaining what went wrong and what the user can do about it.

Feasibility of Implementation

The system should be developed on the basis of readily available libraries and products. These libraries and products should be stable and easily mastered in a short amount of time.

Development Cost

The costs of supporting hardware and software infrastructure should be kept to a minimum.

Maintenance

Reconfiguration of different aspects of the system should be simple and require very little computational overhead.

Updating the system should be made as simple and easy as possible.

Required Models and Specifications

Glossary

An alphabetical list of terms, the glossary includes terms used in the system or the system's description which are either newly introduced, uncommon, or specialized. Each term should be explained in order to create a common language between the requirement analysts, the system architect, the implementation teams and the client.

Use Cases (Requirements Analysis)

Use cases define the interactions between the system and the outside environment. Each use case describes a functional requirement and demonstrates the interaction between an actor (a user or an external system) and the system. It is also possible for use cases to be used to describe a non-functional requirements.

In addition to a basic interaction between the actor and the system, a use case should include the main success scenario and any required pre-conditions and post-conditions. While use cases are a form of text-based requirement documentation, they are best visualised using a system sequence diagram. For more complex use cases (which include several alternatives), use activity diagrams.

Note that acceptance tests should be easily generated using each functional requirement use case.

Class Diagram (Behavioral Analysis)

A class diagram is used to describe the system's structure by presenting its classes. The class diagram should demonstrate the dependencies between the different classes and interfaces of the system, using the relationship notations (such as aggregation, composition, generalization, multiplicity, etc.) and OCL for additional constraints.

Note that the class diagram should reflect the relations elaborated in the different use cases. Additionally, the class diagram should present the main functionality (methods) of each class and show the partition of the classes to different packages.

Component Diagram (System Architecture)

The component diagram represents the way in which the different system components and artifacts interact with each other. Components are autonomous, encapsulated units within a system or subsystem, that provide one or more interfaces. An artifact is a physical unit, such as a file, executable, script, database, etc.

Component diagrams allow to verify that a system's required functionality is being implemented by its components. In addition, component diagrams are useful communication tools for different implementation groups (each group is in charge of a set of components) to visualize the different interaction points between the different components and the different groups.

Deliverables

- Glossary
- Use cases for all the functional requirements, including System Sequence diagrams or Use Case diagrams for every use cases and activity diagram for use cases with several alternatives.
- General description of acceptance test scenarios (generated using the use cases).
- Class diagram
- Component diagram

Class Presentation

Analyze software project management aspects (e.g., requirements management, issue tracking, software configurations management, etc.) and different methodologies to manage each aspect.

Choose an open source tool for project management to use in your project. Examine the methodologies which the tool implements for each of the aspects and explain why this tool best fit your needs.

Please refer to the presentation guidelines at the Assignment page.