

Competitive Matchmaking System

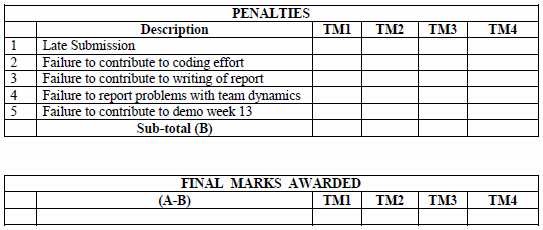
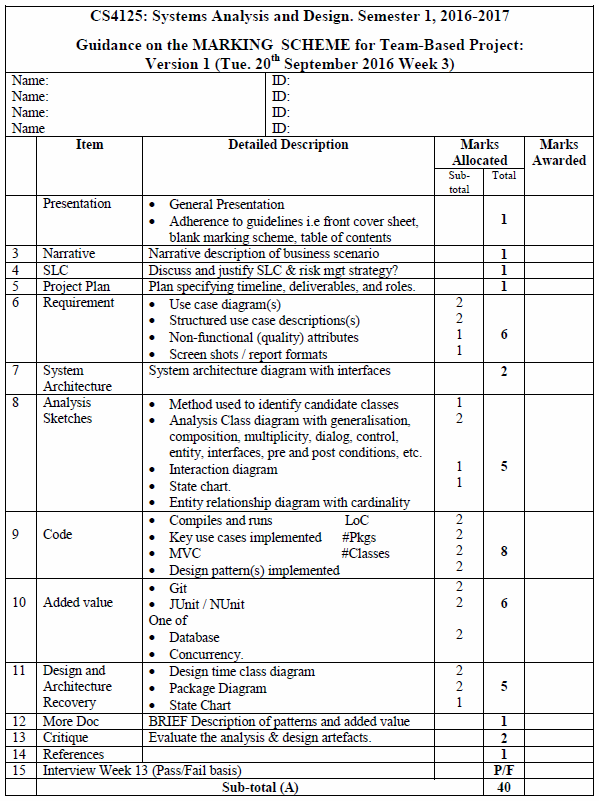
CS4125: System Analysis and Design

Dean Maloney (14140306)

Fergal Bowers (14151324)

Philip Waldron (14173026)

Pawel Pencherzewski (14199041)



# Table of Contents

Narrative Description

Software Life Cycle

Waterfall

V-Model

Agile

Project Plan

Requirements

Use Case Diagrams

Use Case Descriptions

Structured Use Case Descriptions

Non-Functional Requirements

Discussion on Tactics to Support Quality Attributes

Screen Shots

System Architecture

Discussion

Architectural Decisions Taken

UML Workbench

Implementation Language

Architecture Diagram

Analysis Sketches

Identifying Candidate Classes

Class Diagram

Interaction Diagram

State Chart

Entity Relationship Diagram

Code Implementation

Added Value

Git

Junit/NUnit

Database/Concurrency

Design and Architecture Recovery

Design Time Class Diagram

Package Diagram

State Chart

Description of Patterns and Added Value

Critique

References

# Narrative Description

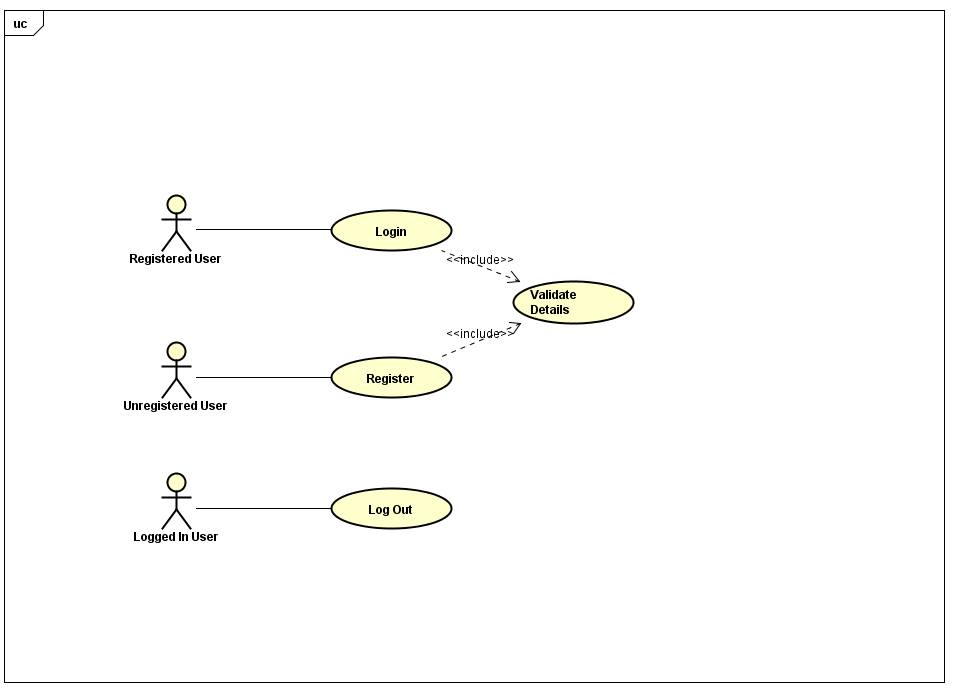
# Software Lifecycle

# Project Plan

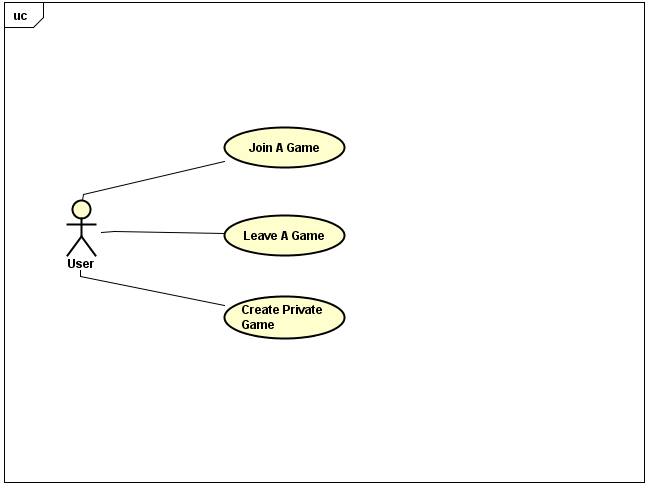
|  |  |  |  |
| --- | --- | --- | --- |
| **Heading** | **Description** | **Allocated to** | **Week** |
| Presentation | Company logo/design cover page | Philip |  |
| Narrative Description | Narrative description of business scenarios | ? |  |
| Software Life Cycle | Discussion of software model used | ? |  |
| Project Plan | Specifying jobs and roles | Philip |  |
| Requirements | Use case diagrams  Use case descriptions  Structured use case descriptions  Non-functional requirements  Tactics to support quality attributes  Screen shots of GUI | Dean  Fergal  Philip, Pawel  ?  ?  ? |  |
| System Architecture | Discussion  Architecture diagram with interfaces | ?  ? |  |
| Analysis Sketches | Identify candidate classes  Class diagrams  Interaction diagram  State chart  Entity relationship diagrams | Group  Dean, Fergal  ?  ?  ? |  |
| Code | Code implementation | Group |  |
| Added Value | Git  JUnit/NUnit  Database/Concurrency |  |  |
| Design and Architecture Recovery | Design time class diagrams  Package diagram  State chart |  |  |
| Patterns and Added Value | Brief description of patterns and added value |  |  |
| Critique | Evaluate the analysis and design artefacts |  |  |
| References | Sources used for learning and information |  |  |

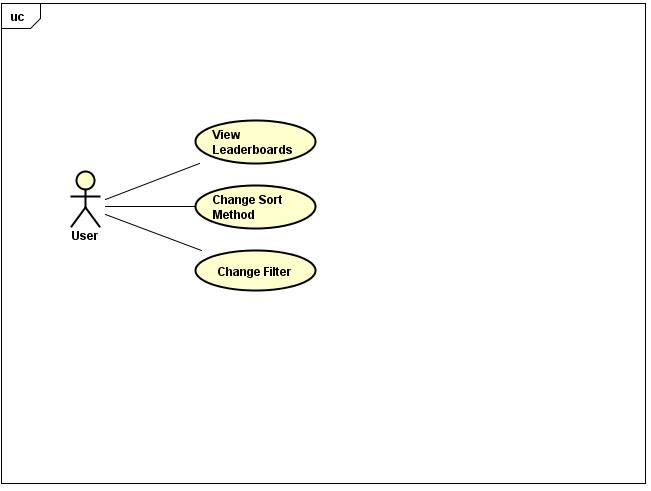
# Requirements

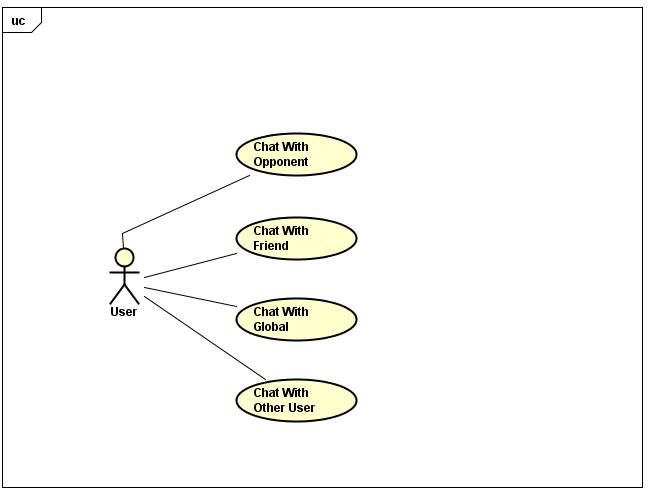
# Use Case Diagrams











# Use Case Descriptions

**Use Case 01:** Chat in Global

**Description:** A user sends a message to the global chat room

**Actors:** User

**Pre-Conditions:**

1. User is logged in.
2. User is connected to the global chat room

**Flow:**

1. User types a message into the text field under global chat.
2. User presses "Send".
3. The Message is send to the chat room.

**Alternate Flow:**

User types a message into the text field under global chat.

1. Request denied: Cannot connect to chat.

**Post-Conditions:**

1. User's message is added to the global chat log.

**Use Case 02:** Chat in Private

**Description:** A user sends a message privately to another player

**Actors:** User, User

**Pre-Conditions:**

1. User is logged in.
2. Other user is online.

**Flow:**

1. User selects other user from playerlist.
2. User selects "Private Message".
3. New chat window is created.
4. User types a message into the text field under private chat.
5. User presses "Send".
6. Message is sent to other user.

**Alternate Flow:**

1. User presses "Send".
2. Request Denied: Other User is offline.
3. New chat window is created.
4. If previous messages were sent between users, populate chat window with past messages

**Post-Conditions:**

1. Users message sent to other user

**Use Case 03:** Chat with Friend

**Description:** A user sends a message to a user on friends list.

**Actors:** User

**Pre-Conditions:**

1. User is logged in.
2. Other User is online.
3. Other user is on user's friends list.

**Flow:**

1. User selects other user from friends list.
2. User selects "Private Message".
3. New chat window is created.
4. User types a message into the text field under private chat.
5. User presses "Send".
6. Message is sent to other user from friends list.

**Alternate Flow:**

1. User presses "Send".
2. Request Denied: Other User is offline.
3. New chat window is created.
4. If previous messages were sent between users, populate chat window with past messages

**Post-Conditions:**

1. User’s message sent to other user from friends list.

**Use Case 04:** Chat in lobby

**Description:** A user sends a message to all users in game lobby

**Actors:** User

**Pre-Conditions:**

1. User is logged in.
2. User is in a game lobby.

**Flow:**

1. User types a message into the text field under lobby chat.
2. User presses "Send".
3. The Message is send to the lobby chat room.

**Alternate Flow:**

User types a message into the text field under lobby chat.

1. Request denied: Cannot connect to chat.

**Post-Conditions:**

1. User's message is added to the lobby chat log.

# Structured Use Case Descriptions

|  |  |  |
| --- | --- | --- |
| **USE CASE 05** | View Leaderboards | |
| **Goal in Context** | The leaderboards are displayed to the user in another window | |
| **Scope & Level** |  | |
| **Preconditions** | None | |
| **Success End Conditions** | The leaderboards are displayed | |
| **Failed End Condition** | The leaderboards are not displayed | |
| **Primary, Secondary, Actors** | User | |
| **Trigger** | ViewLeaderboards request comes in | |
| **DESCRIPTION** | **Step** | **Action** |
| 1 | User selects View Leaderboards button |
| **EXTENSIONS** | **Step** | **Branching Action** |
| 1a | User wants to view leaderboards:  1a. A new window opens, populated with all registered users |

|  |  |  |
| --- | --- | --- |
| **USE CASE 06** | Change Sort Method | |
| **Goal in Context** | The leaderboards are displayed in a desired sorted order | |
| **Scope & Level** |  | |
| **Preconditions** | Leaderboards are currently displayed | |
| **Success End Conditions** | Sort method is changed | |
| **Failed End Condition** | The sort method is not successfully changed | |
| **Primary, Secondary, Actors** | User | |
| **Trigger** | ChangeSortMethod(sort) request comes in | |
| **DESCRIPTION** | **Step** | **Action** |
| 1 | User selects the method in which to sort the leaderboards |
| **EXTENSIONS** | **Step** | **Branching Action** |
| 1a | Sort method changed:  1a. The leaderboards are repopulated in the relevant order |
| **VARIATIONS** | **Step** | **Branching Action** |
| 1 | User may sort by:  Alphabetical order  Most wins  Highest win/loss ratio |

|  |  |  |
| --- | --- | --- |
| **USE CASE 07** | Change Filter | |
| **Goal in Context** | A filter is applied and the leaderboards are displayed taking into account the new filter | |
| **Scope & Level** |  | |
| **Preconditions** | Leaderboards are currently displayed | |
| **Success End Conditions** | Leaderboards are displayed in relation to the filter | |
| **Failed End Condition** | Leaderboards are not filtered correctly | |
| **Primary, Secondary, Actors** | User | |
| **Trigger** | ChangeFilter request comes in | |
| **DESCRIPTION** | **Step** | **Action** |
| 1 | User types in a filter |
| 2 | User selects a button to apply the filter |
| **EXTENSIONS** | **Step** | **Branching Action** |
| 2a | Filter is applied:  1a. The leaderboards are repopulated taking to account the filter |

|  |  |  |
| --- | --- | --- |
| **USE CASE 08** | Add a Friend | |
| **Goal in Context** | Causes a pop up window to appear where the user can enter a name of another user to add as a friend. | |
| **Scope & Level** |  | |
| **Preconditions** | We know: adding user and the user being added | |
| **Success End Conditions** | One user successfully sent a friend invite, the other received it | |
| **Failed End Condition** | The invitation has not been sent/ the end user has not received an invitation | |
| **Primary, Secondary, Actors** | User sending and user receiving a friend invitation | |
| **Trigger** | AddFriend request comes in | |
| **DESCRIPTION** | **Step** | **Action** |
| 1 | User sends a friend invitation by entering a name |
| 2 | The other user receives a friend invitation through “respond to invite” tab |
| 3 | Receiving user can accept or decline the invitation |
| **EXTENSIONS** | **Step** | **Branching Action** |
| 1a | A user does not exist:  1a1. Display error and give another try |

|  |  |  |
| --- | --- | --- |
| **USE CASE 09** | Remove a friend | |
| **Goal in Context** | Causes a pop up window to appear where the user can enter a name of a user he/she wants to remove from friends list and him/herself from the other user’s f/l. | |
| **Scope & Level** |  | |
| **Preconditions** | We know: the user removing and the user being removed | |
| **Success End Conditions** | A user successfully removed another user from their friends list and removed himself from their f/l. | |
| **Failed End Condition** | The user entered to be removed from the friends list has not been removed. | |
| **Primary, Secondary, Actors** | User removing a friend and the friend being removed | |
| **Trigger** | RemoveFriend request comes in | |
| **DESCRIPTION** | **Step** | **Action** |
| 1 | User enters a name that he/she wishes to remove from friends list. |
| 2 | Both users’ friends list are updated, removing each other from them. |
| **EXTENSIONS** | **Step** | **Branching Action** |
| 1a | A user does not exist:  1a1. Display error and allow another try |

|  |  |  |
| --- | --- | --- |
| **USE CASE 10** | Invite to a Game | |
| **Goal in Context** | Causes a pop up window to appear where he/she can enter a name of another user to send a game invite to. | |
| **Scope & Level** |  | |
| **Preconditions** | We know: inviting user and the user being invited | |
| **Success End Conditions** | One user successfully sent a game invite, the other received it | |
| **Failed End Condition** | The invitation has not been sent/ the end user has not received an invitation | |
| **Primary, Secondary, Actors** | User sending and user receiving the game invitation | |
| **Trigger** | InviteToGame request comes in | |
| **DESCRIPTION** | **Step** | **Action** |
| 1 | User sends a game invitation by entering a name |
| 2 | The other user receives a game invite through “respond to invite” tab |
| 3 | Receiving user can accept or decline the invitation |
| **EXTENSIONS** | **Step** | **Branching Action** |
| 1a | A user does not exist:  1a1. Display error and allow another try |
| 1b | A user is not online:  1b1. Display error and allow another try |

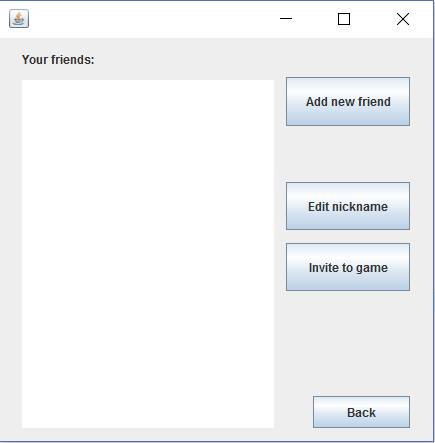
|  |  |  |
| --- | --- | --- |
| **USE CASE 11** | Respond to Invite | |
| **Goal in Context** | The user is displayed a list of friend and game invitations, which he/she can accept, decline or ignore. | |
| **Scope & Level** |  | |
| **Preconditions** | We know: the user, his/her invitations | |
| **Success End Conditions** | All invitations are displayed correctly and the user is able to respond to them successfully.  Responding to an invite removes it from the list. | |
| **Failed End Condition** | The invitations don’t display or are displayed incorrectly.  The invitations do not respond to actions taken. | |
| **Primary, Secondary, Actors** | The user viewing the invitations | |
| **Trigger** | RespondToInvite request comes in | |
| **DESCRIPTION** | **Step** | **Action** |
| 1 | User presses a Respond to Invite tab/button. |
| 2 | A list of invitations for this user is displayed |
| 3 | Accepting/declining an invitation updates the invitation list. |
| 4 | If user accepted friend invitation the friend lists of both users are updated |
| 5 | If user accepted a game invite he/she is sent to a game screen. |
| **EXTENSIONS** | **Step** | **Branching Action** |
| 5a | User sending invitation is offline:  5a1. Display information that user is offline |

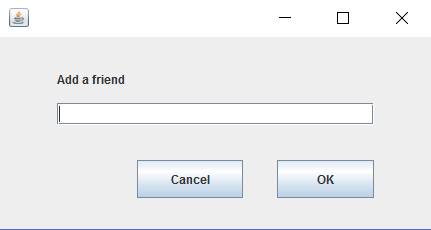
|  |  |  |
| --- | --- | --- |
| **USE CASE 12** | Nickname Friend | |
| **Goal in Context** | User adds a nickname beside another friend’s username in his/her friend list. | |
| **Scope & Level** |  | |
| **Preconditions** | We know: user adding nickname, the names on his f/l | |
| **Success End Conditions** | User successfully appended a nickname to a friend’s username in his/her friend list. | |
| **Failed End Condition** | No nickname is displayed beside the friend that the nickname was added to. | |
| **Primary, Secondary, Actors** | The user adding the nickname | |
| **Trigger** | AddNickname request comes in | |
| **DESCRIPTION** | **Step** | **Action** |
| 1 | User presses Nickname a friend button |
| 2 | User types in the name of a friend he wants to add a nickname to |
| 3 | User accepts changes made |
| **EXTENSIONS** | **Step** | **Branching Action** |
| 2a | A user does not exist:  2a1. Display error and allow another try |
| 3a | The nickname entered is too long:  3a1. Display error and exit “nicknaming” action |

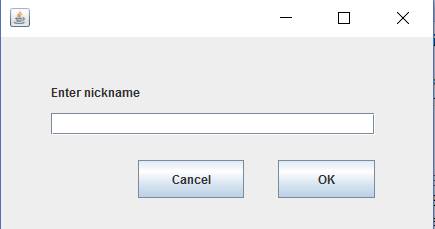
|  |  |  |
| --- | --- | --- |
| **USE CASE 6** | View Friend Availability | |
| **Goal in Context** | User is displayed with a list of his/her friends, as well as their online status. | |
| **Scope & Level** |  | |
| **Preconditions** | We know: A user, his/her friends and their online statuses | |
| **Success End Conditions** | The friends are correctly displayed with correct online statuses for every individual user | |
| **Failed End Condition** | The friends are displayed incorrectly or not at all.  The online statuses are incorrect | |
| **Primary, Secondary, Actors** | User viewing his/her friends list. | |
| **Trigger** | ViewFriends request comes in | |
| **DESCRIPTION** | **Step** | **Action** |
| 1 | User presses View friends button/tab. |
| 2 | A list of all friends is displayed to the user with correct online statuses |

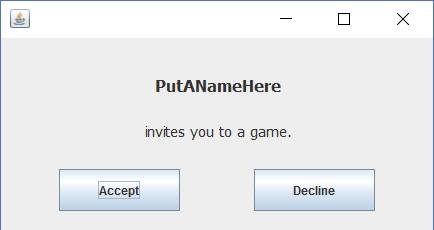
# Discussion on Tactics to Support Quality Attributes

# Screen Shots





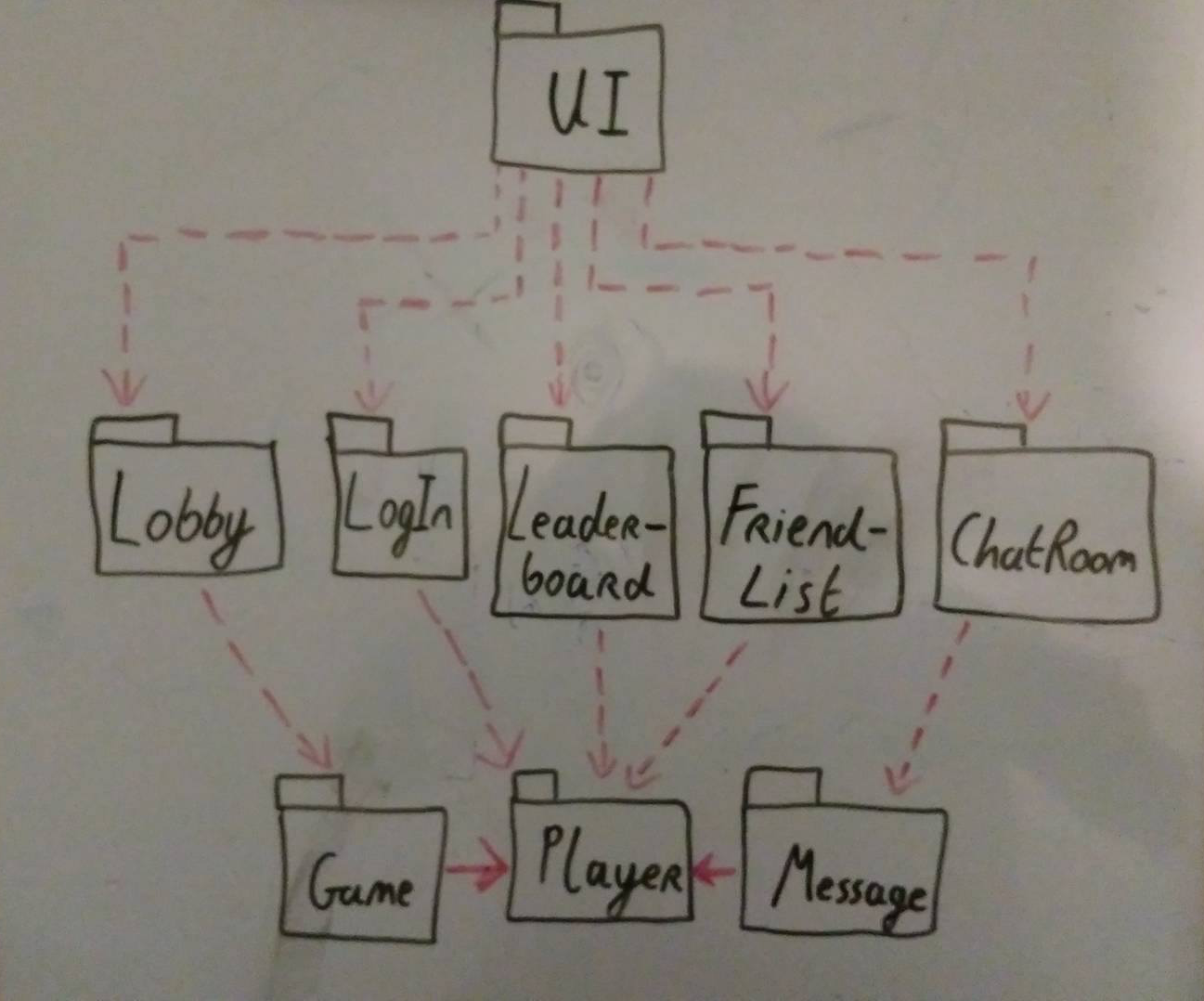




# System Architecture

# Discussion

# Architecture Diagram



# Analysis Sketches

# Identify candidate classes

Using a data driven design method, we came up with a list of potential candidate classes by using the noun identification technique.

Initial list:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Username | Password | Registration | Logging | Game | System |
| Play | Invite | Message | Profile | User | Notification |
| Request | Send | Accept | Decline | User Interface | Display |
| Friend | Friendlist | Dialog | Input | Leaderboard | Lobbylist |
| Chatroom | Player | User | Text | Connection |  |

Heuristics:

1. Too vague or to specific – RED
2. An operation – BLUE
3. An attribute - Orange
4. Out of Scope - GREEN
5. Other similar objects – Purple

Filtered list:

|  |  |  |  |
| --- | --- | --- | --- |
| User | Player | Game | Leaderboard |
| Lobbylist | Friendlist | Friend | Chatroom |
| Message | Invite | User Interface | Log |

## User

A user encompasses all the data related to the logged in user who uses the application. Their data is updated after each game they complete.

## Player

A user who is matched into a game becomes a player, accepting their inputs to play the game.

## Game

The game is the logic for the game to be played.

## Leaderboard

The leaderboard encompasses all the data to be displayed on the leaderboard, letting data displayed be sorted by various sorting methods.

## Lobbylist

Lobbylist keeps track of all open connections, filtering out unavailable connections (already in game, etc.).

## Friendlist

Friendlist keeps track of all the added friends of the current user.

## Friend

Friend encompasses the data of a friend of the user, allowing certain elements to be edited such as a nickname.

## Chatroom

Chatroom handles messages that the user receives from other users, friends, the server etc. and allows the user to input messages to be sent to their desired destination.

## Message

A message encompasses the data relevant to a sent message; its contents, sender and destination.

## Invite

There are two types of invites:

1. Game invite: Allows a user to challenge a friend directly to play a game, and allows the recipient to choose to accept.
2. Friend invite: Allows a user to send a friend request to another user, and allows the recipient to choose to accept.

## UI

Handles all graphical elements that the user may interact with once they launch the application.

## Log

Log is for taking input information from the user. It encompasses users logging in, out and registering.

# Class diagrams

# Interaction diagram

# State chart

# Entity relationship diagrams

# Code Implementation