# SERENITY GAMING

CS4125: System Analysis and Design Project

SEMESTER 1 - 2014

GROUP NAME: TIGER LILLY ORANGE

W
X
Y
Z

## CS4125: Systems Analysis and Design. Semester I, 2014-2015

**Guidance on the MARKING SCHEME for Team-Based Project: Version 1 (24th September 2014 - Week 3)**

| | |
|---|---|
| Name: | ID: |
| Name: | ID: |
| Name: | ID: |
| Name | ID: |

| | Item | Detailed Description | Marks Allocated | | Marks Awarded |
|---|---|---|---|---|---|
| | | | Sub-total | Total | |
| | Presentation | • General Presentation<br>• Adherence to guidelines i.e. front cover sheet, blank marking scheme, table of contents | | **2** | |
| 3 | Narrative | Narrative description of business scenario | | **1** | |
| 4 | SLC | Discuss/justify SLC & risk management strategy? | | **1** | |
| 5 | Project Plan | Plan specifying timeline, deliverables, and roles. | | **1** | |
| 6 | Requirement | • Use case diagram(s)<br>• Structured use case descriptions(s)<br>• Non-functional (quality) attributes<br>• Screen shots / report formats | 1<br>2<br>2<br>1 | **6** | |
| 7 | System Architecture | System architecture diagram with interfaces | | **2** | |
| 8 | Analysis Sketches | • Method used to identify candidate classes<br>• Class diagram with generalisation, composition, multiplicity, dialog, control, entity, interfaces, pre and post conditions, etc.<br>• Interaction diagram<br>• Entity relationship diagram with cardinality | 1<br>2<br>1<br>1 | **5** | |
| 9 | Code | • Compiles and runs<br>• MVC<br>• Design pattern AND/OR Concurrency. A bonus of 4 marks will be awarded where both design pattern(s) and concurrency implemented | 2<br>3<br>4 | **9** | |
| 10 | Design blueprints based on code | • Architectural diagram<br>• Class diagram □ Interaction diagram □ State chart.<br>• Description of patterns and approach to concurrency support. | 1<br>2<br>1<br>2<br>1 | **7** | |
| 11 | Critique | Evaluate the analysis & design artefacts. | | **2** | |
| 12 | References | | | **1** | |
| | Online Assessment | Week 6: Use cases<br>Week 7: Class diagram<br>Week 9: Code demo | 1<br>1<br>1 | **3** | |
| | Interview Week 13 (Pass/Fail basis) | | | **P/F** | |
| | **Sub-total** | | | **40** | |
| | **Bonus (max of 4)** | | | | |
| | **Total** | | | | |

# Table of Contents

# Description

## Overview

As the online gaming market continues to grow yearly (having grown from a 7.4 billion dollar market in 2003 to a 41.4 billion projected estimate for 2015), the need to cater for its community is ever more prevalent. We assert that a core aspect of failure of breakthrough for new online games is the lack of community management undertaken by the publishing company. Most companies treat the community as an organism that will cater for itself, relying on fan made forums as the go to hub for their game with the thought process that this is how other already successful companies' communities function.

However those already successful companies' communities grew over many years in the industry. This is a luxury new titles do not have, and so we believe companies should begin outsourcing community management to a game hosting service that can keep all their players together in a single place through a user friendly, feature packed service that bolsters community building.

The finest example of community necessity for online games comes in the MMORPG sector (but this can be seen in any online genre), where none can argue that World of Warcraft (WoW) dominates the market. Yet consistently new titles for this genre are announced and greeted with massive anticipation (Star Wars – The Old Republic is the finest example), all these titles are almost immediately dubbed "WoW Killers", yet over and over again they die after a few months and usually have to completely redesign monetary plans, design plans etc.

The constant factor in these failures is the lack of community. Players come over from established titles like WoW but are thrown into the game with maybe a few friends, but no direct access to any form of community for the new title. Also at launch for these games there is often a dozen or more fan sites/community hubs already generated but this is just splitting of players even further. The end result is all too common.

That is where our company Serenity Gaming comes in. We aim to provide users a means of access to the companies' game where they can also interact with their friends, clan members, and the whole gaming community. All game news, patches, events, competitions etc. will also be available directly from our client service. All of this allows companies to simply focus on their game development as they always would and produce the best possible game, while we handle the community features that in the long run are going to create the community that is going to give true longevity to the publisher's franchise.

## What is Serenity Gaming?

Serenity Gaming is a client program that gives players access to a company's game and acts as the games root community hub. It gives direct access to friends list, clan list, clan TeamSpeak (voice over IP), direct link to game and community forums, direct access to all game updates, news and events.

Players play the game directly through Serenity Gaming which handles matchmaking, leaderboards, game lobbies and gameplay stats. Gameplay stats and leaderboards are visible to the player at any time through the client, they can also check global, friends and clan member stats as they wish.

# Software Lifecycle Model

When considering software lifecycle models such as the waterfall model and V-Model we found that both would not satisfy our project needs for a variety of reasons.

## Waterfall Model:

- No adaptability to requirements being re-defined by client. Any change in requirements after design was finalized would require the design being modified to accommodate the changes. This effectively means invalidating a good deal of working hours which means increased cost.
- We may not be aware of future implementation difficulties when writing the design for Serenity Gaming. I.e. it may become clear in the implementation phase that a particular area of the programs functionality is extraordinarily difficult to implement. In such case we need a lifecycle mode that allows for revision of the design rather than persist in a design based on faulty predictions.
    - David Parnas, writes in "A Rational Design Process: How and why to Fake it":
        - "Many of the [system's] details only become known to us as we progress in the [system's] implementation. Some of the things that we learn invalidate our design and we must backtrack."

## V-Model:

- Inflexible, encourages a rigid and linear view of software development and has no inherent ability to respond to change.
- It provides only a slight variation to the waterfall model and is therefore subject to the same criticisms as that model.
- It is consistent with and therefore implicitly encourages, inefficient and ineffective testing methodologies. It implicitly promotes writing test scripts in advance rather than exploratory testing; it encourages testers to look for what they expect to find, rather than discover what is truly there. It also encourages a rigid link between the equivalent levels of either leg, rather than encouraging testers to select the most effective and efficient way to plan and execute testing.

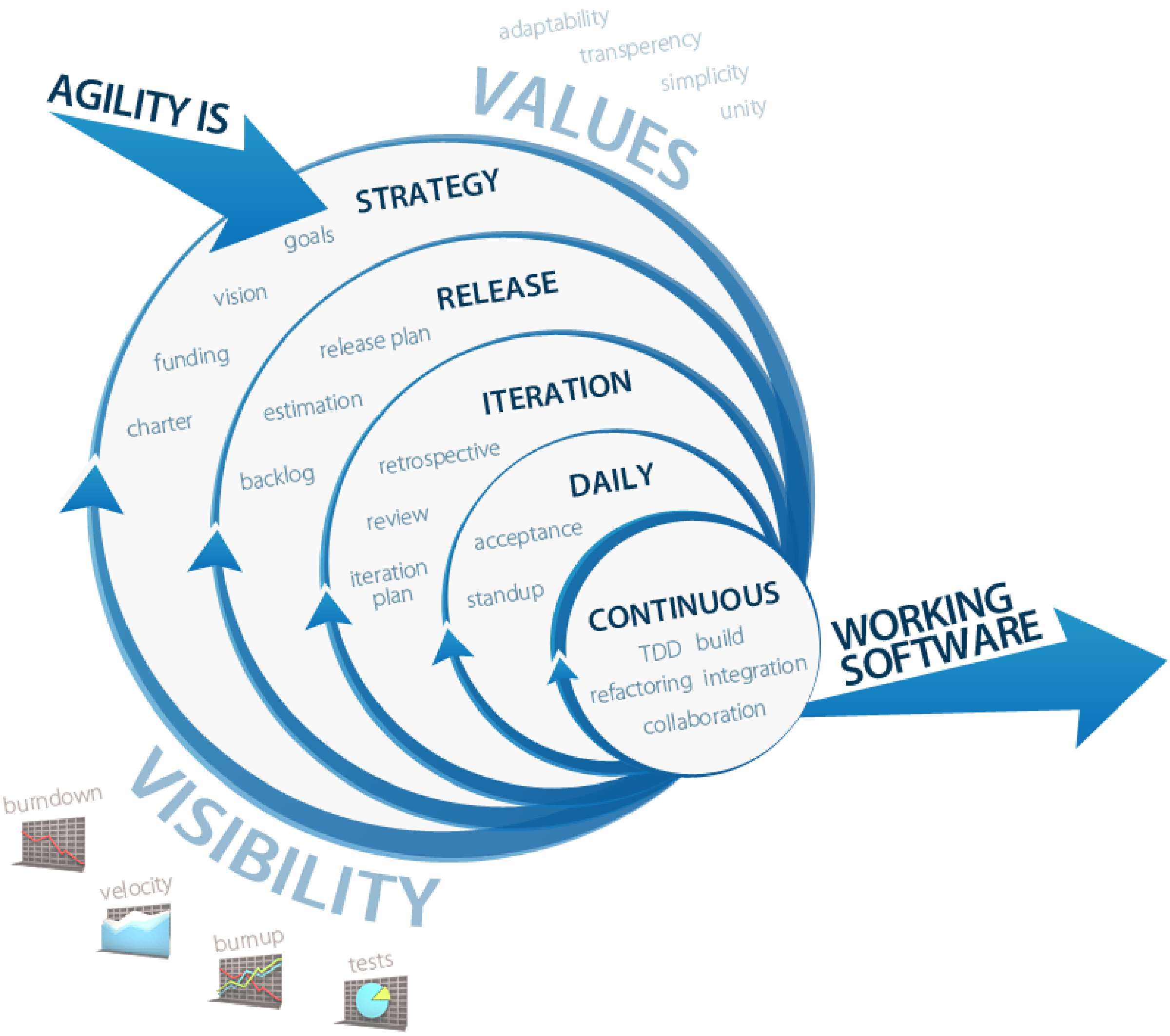
## Agile Software Development Model:

With the above criticisms in mind we aimed to find a more appropriate approach, keeping adaptability in mind as our core focus. This being a game hosting service responding to change will be a massive feature. When working in conjunction with a game publishing company, change to specification will be a frequent factor. We also needed to be prepared for future growth of the title franchise with the need to accommodate frequent game updates and expansions launches.

That being said we elected to work under the Agile Software Development Model. It allows us to take an iterative and incremental approach to development. We aim to implement continuous improvements into our service as part of the game publisher's needs and our own design visions to constantly improve our service to meet user needs and give them the best possible service. All with the goal of creating a seamless community for the players that will improve success of the game title.

A Diagram is included on the next page to demonstrate our approach of development under the Agile Software Development Model.

Agile Model Diagram



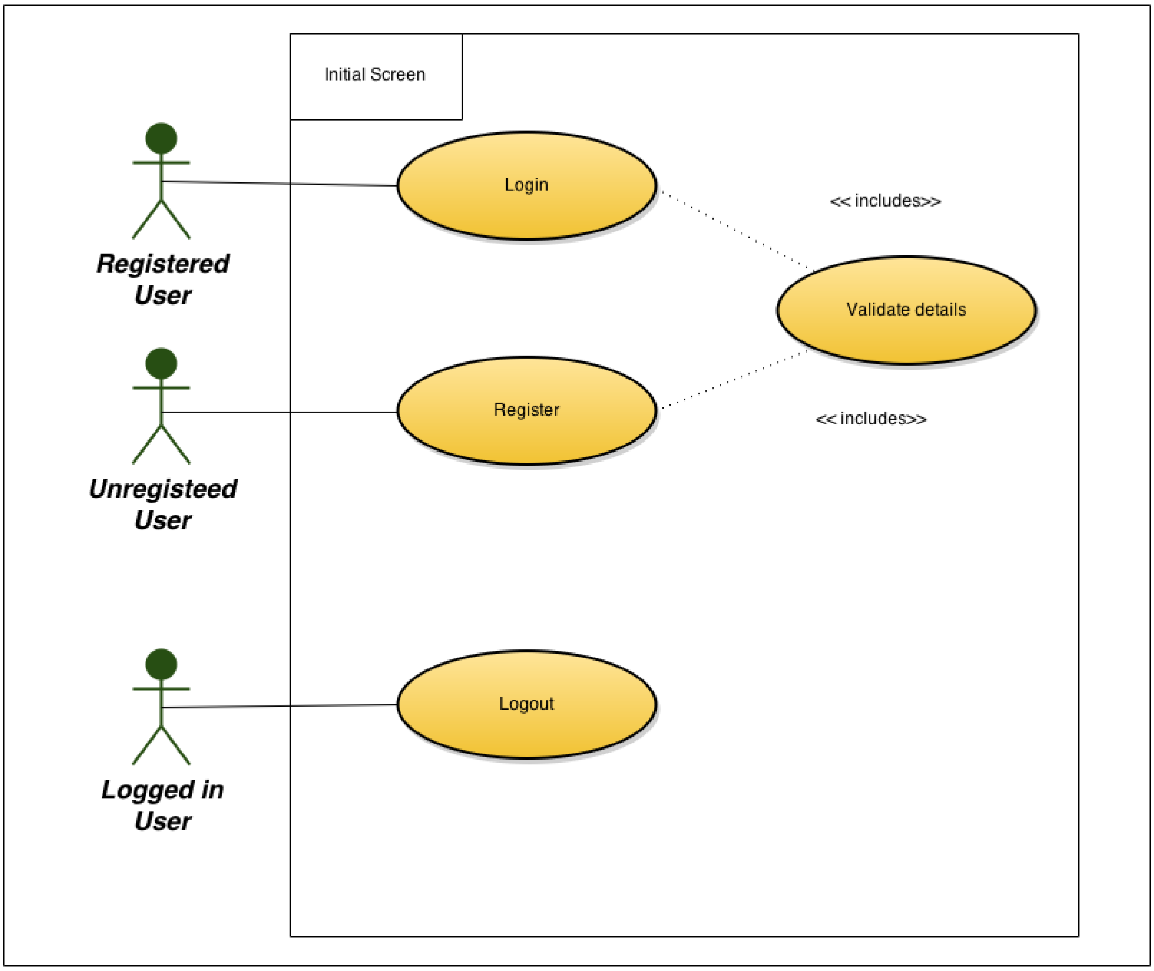
## Project Plan and Allocation of Roles

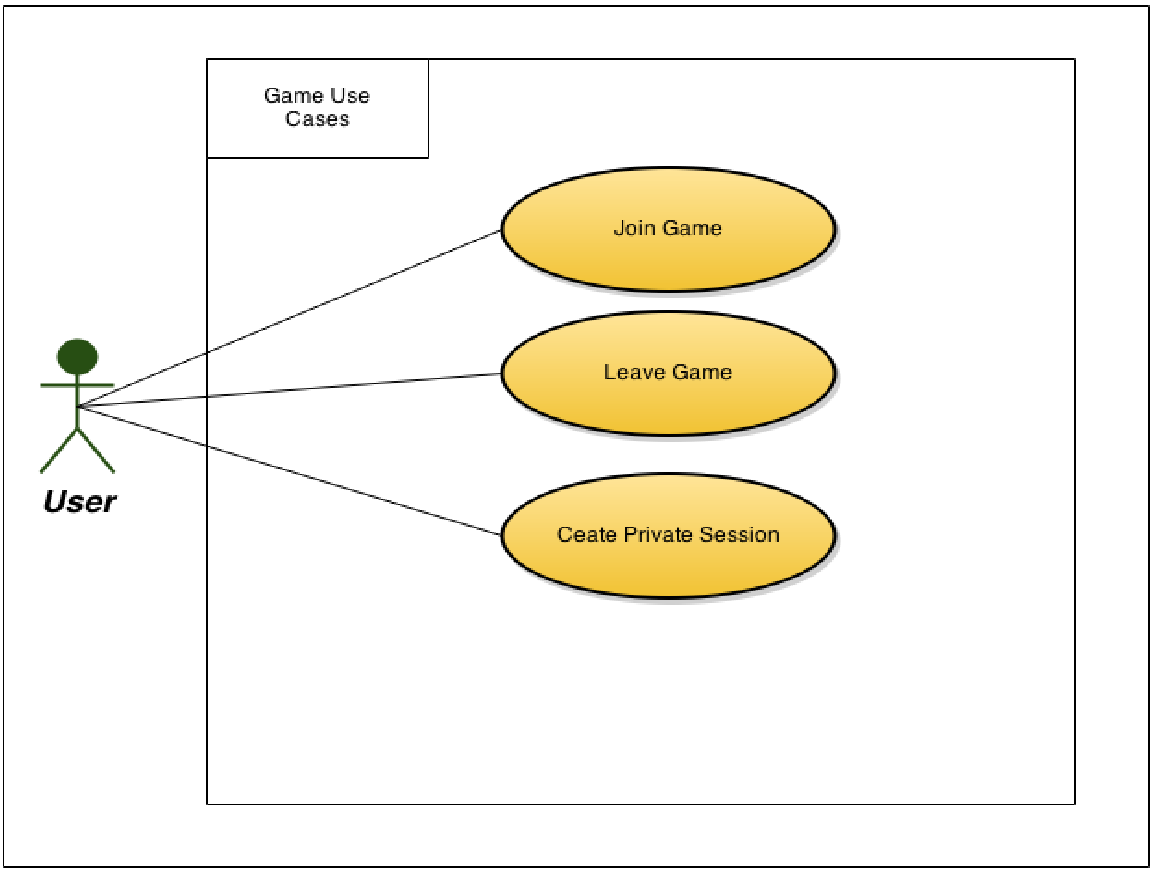
| Deliverable | Description | Person | Week |
|---|---|---|---|
| Presentation | Company Logo/Design doc cover page | | 8 |
| Narrative | Narrative Description of business scenarios | | 4 |
| Software Lifecycle | Discussion of the software model used. | | 9 |
| Project Plan | Specifying timeline, deliverables, and roles | Group-Allocation -Writeup | 5 |
| Requirements | Use case diagrams | Group | 7 |
| | Use case descriptions | | 6 |
| | Structured Use Case Descriptions | | 8 |
| | Non-functional (quality attributes) | | 10 |
| | Screen shots | | 12 |
| System Architecture | Architecture diagram with interfaces | | 7 |
| Analysis Sketches | Identify Classes | | 6 |
| | Class Diagram | | 7 |
| | Communication Diagram | | 8 |
| | E-R diagram | | 8 |
| Code | Code Implementation of Serenity Gaming | Group | 8 - 12 |
| Design | Architectural Diagram | | 12 |
| | Class Diagram | | 12 |
| | Interaction Diagram | | 12 |
| | State chart | | 12 |
| | Description of patterns, approach to concurrency support | | 12 |
| Critique | Critique on quality of analysis/design | | 12 |
| References | Sources used for learning/information | | 4 - 12 |

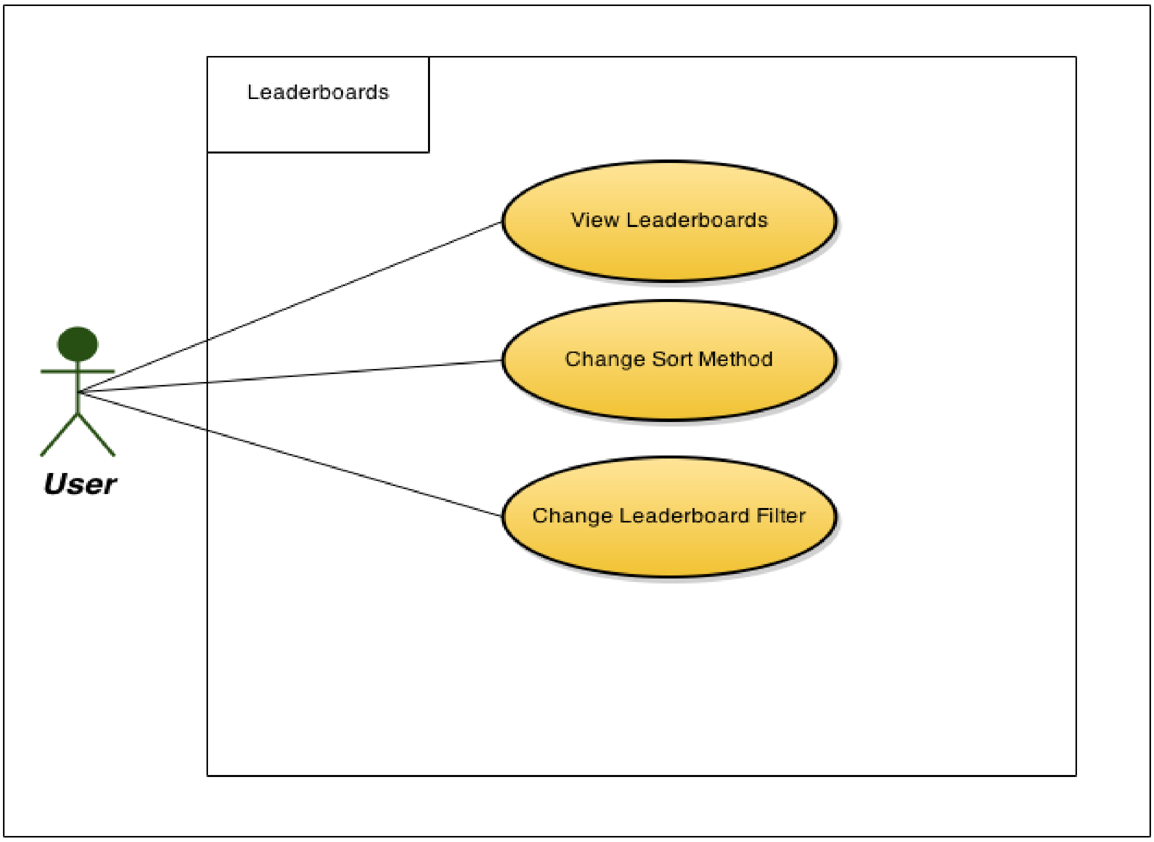
# Requirements

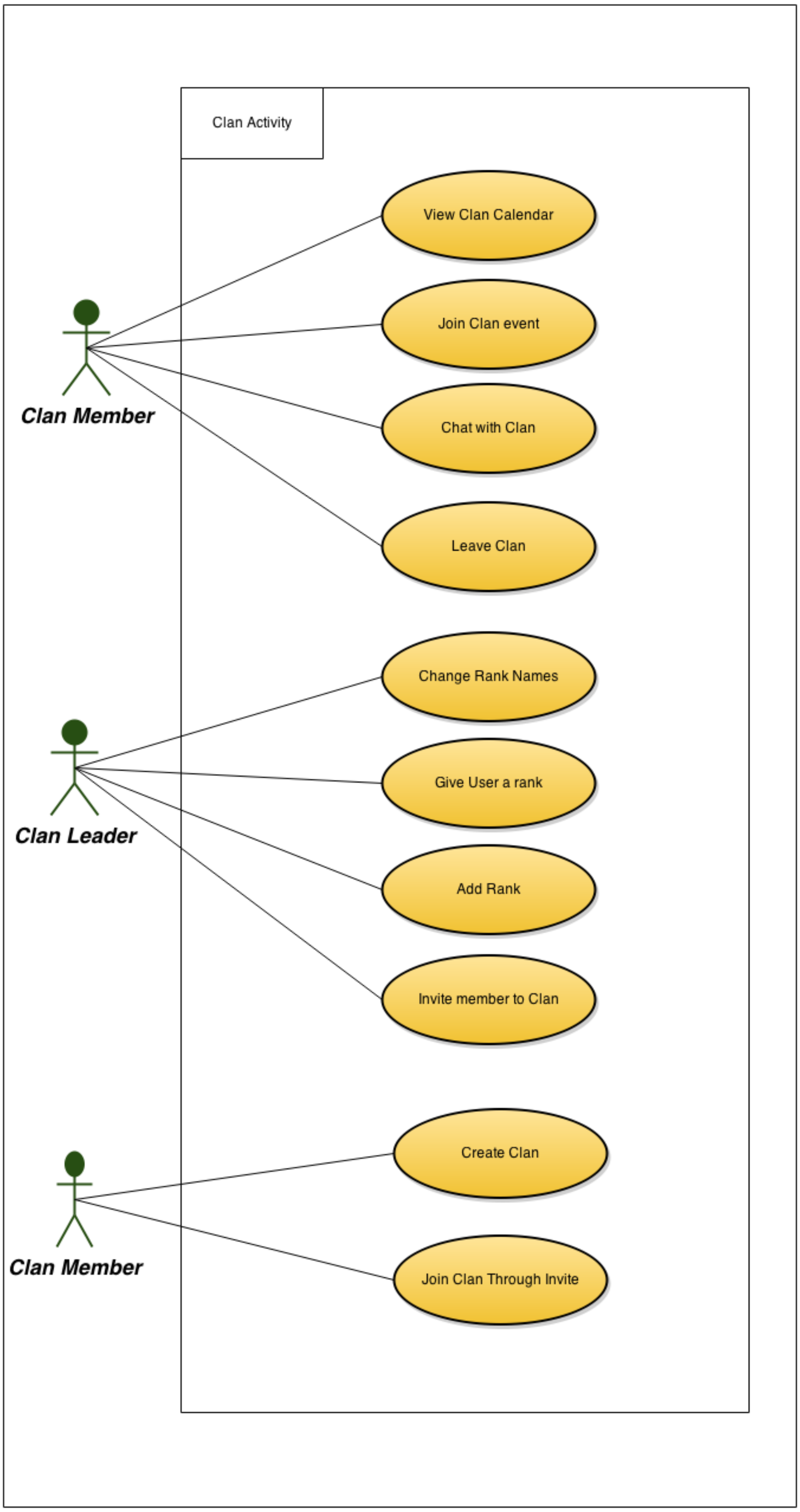
## Functional Requirements

### Use Case Diagrams




Game Use Cases
Join Game
Leave Game
Ceate Private Session
User


Leaderboards
View Leaderboards
Change Sort Method
Change Leaderboard Filter
User


Clan Activity
Clan Member
View Clan Calendar
Join Clan event
Chat with Clan
Leave Clan
Clan Leader
Change Rank Names
Give User a rank
Add Rank
Invite member to Clan
Clan Member
Create Clan
Join Clan Through Invite

# Detailed Use Case Descriptions

## Join Game

**Name**: Join Game

**Description**: A user joins a game.

**Actors**: User

**Pre-Conditions**:

1. User is logged in.

**Flow**:

1. User selects “join game”.
2. User’s profile enters matchmaking.
3. User is entered into a suitable lobby.
4. Countdown started.
5. User can modify avatar characteristics.
6. Timeout.
7. Game is launched.

**Alternate Flow**:

8. User selects “join game”.
    a. Request denied: player banned.
    b. Invite friend(s) to game.
9. Countdown started.
    a. If game is already underway, move user to customise character screen.

**Post-Conditions**:

1. The user is added to the game.

## Clan Creation

**Name:** Clan Creation

**Actors:** An Active User who is not currently in a clan

**Description:** Active User can create a new clan in system. They must supply valid information for creation and then become the Clan leader and are given the Clans unique Forum address and TeamSpeak server address.

**Pre-Condition:**

**1)** Actor must be a registered user of game hosting service

**2)** Actor must not currently be a member of existing clan.

**3)** Clan name must be unique and not violate terms of conduct.

**Post Condition:**

**1)** Unique Server address for new clan must be created

**2)** Unique Forum address for new clan must be created

**3)** Active User account must be registered in system as Clan Leader/Member

**Flow:**

1. User logs into system
2. User selects create new clan option
3. User account is checked for clan member status:
   a. Already in clan: Display error and return from creation.
4. Creation form is presented to user.
5. User fills in: Clan name (Required), Password (Required), Clan Logo(Optional), Rank titles (1-4) (Optional)
6. System reads and confirms submitted information.
   a. Clan name in use: Display error and return to 5.
   b. Information submitted against naming policies (abusive language etc.): Display error and return to 5.
7. Assign Active user as Clan leader and update Active user account in system to be recognised as clan leader.
8. Display success message to Active user and return from creation.
9. Generate unique server and forum addresses for Clan Leader.
10. Send generated information to Clan Leader account email.

## Assign User a Rank

**Name:** Assign User a Rank

**Actors:** User

**Description:** A user assigns a clan member a rank.

**Pre-Condition:**

1. User must have permission to change members ranks (i.e. is a Clan Leader)
2. Member whose rank is being changed must not be banned.

**Post Condition:**

1. Members rank must be updated

**Flow:**

1. User selects change clan members rank
2. User selects the member they wish to the change rank of.
    a. Member is banned, rank can't be changed.
3. System displays change rank screen.
4. User selects the new rank to be assigned to the member
5. Members rank is updated

## Discussion on Tactics to Support Quality Attributes

We decided to use Java as the language to write our program in after much discussion. Java allows portability across many architectures which is a major benefit. Java is also the most used language by programmers worldwide. This would make recruiting new staff to work on our system easier, as most software professionals are familiar with the language.

Also our client rendering demands are quite low. While we aim to provide an aesthetically pleasing service, we are not developing graphically demanding software where a language such as C++ would be preferable for speed, referring to rendering speed here rather than response time which is still an important factor for our software.

We wanted to make our UI user friendly and accessible. Users new to our system should be able to navigate through our menus and windows intuitively. We can support this by having a clear, straight forward layout to our user interface, keeping it simplistic.

We looked into Heartbeat Monitor to monitor our server's health. This would help improve our systems reliability. We would also have a number of back up databases to protect the integrity of our data.

## Structured Use Case Descriptions:

### Register

| Actor Action | System Response |
|---|---|
| 1. Selects "Register" | 2. Register form displayed |
| 3. Enters details and clicks submit | 4. Registration details are verified, user is sent verification email |
| Alternative course:<br>3a: Invalid information entered:<br>• System displays error regarding incorrect information | |

Non Functional Requirement: Security
System should encrypt the user's personal information and passwords and store them securely.

## Login

| Actor Action | System Response |
|---|---|
| 1. Selects ” Login” | 2. Login form displayed |
| 3. Enters details and clicks submit | 4. Login details are verified, game dashboard displayed |
| Alternative course:<br>3a: Invalid information entered:<br>• System displays error regarding incorrect information | |

Non Functional Requirement: Performance
System should respond within 5 seconds to the login request.

## Logout

| Actor Action | System Response |
|---|---|
| 1. Selects “Logout” | 2. System Updates and saves their session, user is set to offline |

## Create clan

| Actor Action | System Response |
|---|---|
| 1. Selects “Create Clan” | 2. Presents Clan Creation form |
| 3. Enters details and clicks submit | 4. Unique server and forum web address sent to user |
| Alternative course:<br>1a: Already in a clan:<br>- Create clan fails<br>3a: Invalid information entered:<br>- System displays error regarding incorrect information | |

Non Functional Requirement: Security
System should encrypt the clan’s information and store it securely.

## Invite Member to clan

| Actor Action | System Response |
|---|---|
| 1. Selects “Invite Member” | 2. Displays screen with a text field to enter username you wish to invite |
| 3. Enters username of member to invite | 4. Sends invite to the user, and sends email to their address notifying of the invite. User is added to pending invites list of the clan. |
| Alternative course:<br>3a: Username entered isn’t valid<br>- System displays error regarding incorrect username | |

## Join Clan (through Invitation)

| Actor Action | System Response |
|---|---|
| 1. Clicks “View Invites” | 2. Displays clan invites a user has |
| 3. Selects invite from clan they wish to join, and clicks “Accept” | 4. User is added to the clan, clan is updated with the user added and users profile is updated to having joined the clan |

## Leave clan:

| Actor Action | System Response |
|---|---|
| 1. Selects “Leave clan” | 2. Displays message asking for confirmation of decision |
| 3. Clicks “Leave” | 4. Clan is updated with user removed, users profile is updated to reflect they are no longer in a clan |

## Assign User a Rank:

| Actor Action | System Response |
|---|---|
| 1. Selects “Assign User Rank” | 2. Displays list of users with their ranks |
| 3. Click on member they wish to change rank of | 4. Displays members profile with options menu |
| 5. Clicks “Change Rank” from options menu | 6. Displays list of ranks that can be changed to. |
| 7. Clicks on desired rank | 8. Members profile updated with new rank |
| Alternative course:<br>3a. User selected is banned<br>• Cannot change rank, system alerts the user to the players banned status. | |

## Add Rank:

| Actor Action | System Response |
|---|---|
| 1. Selects “Add Rank” | 2. Displays Add Rank Screen with form |
| 3. Enters new ranks details | 4. Updates clan with new rank added |
| Alternative course:<br>3a. Rank name entered is already used for a different rank<br>• Cannot change rank name, System displays error. | |

## Change Rank Name:

| Actor Action | System Response |
|---|---|
| 1. Selects “Change Rank Name” | 2. Displays list of current ranks |
| 3. Selects the rank they wish to edit | 4. No action |
| 5. Enters new name or rank | 6. Clan updated with new rank name |
| Alternative course:<br>3a. Rank name entered is already used for a different rank<br>• Cannot change rank name, System displays error. | |

## Create event:

| Actor Action | System Response |
|---|---|
| 1. Selects “Create Event” | 2. Displays event creation form |
| 3. Enters event details and submits | 4. Event is created, and clan is updated with new event added to the calendar |
| | |

## Accept Event Invite

| Actor Action | System Response |
|---|---|
| 1. Selects “Event Invites” | 2. Displays users active event invites |
| 3. Selects invite and clicks accept | 4. User added to event, event participates list updated |
| | |

## View Leaderboards

| Actor Action | System Response |
|---|---|
| 1. Selects “View Leaderboards” | 2. Displays Leaderboards screen |

## Changes Leaderboard Sort Method:

| Actor Action | System Response |
|---|---|
| 1. Clicks on column the user wishes to sort the leaderboard by | 2. Displays the leaderboard sorted by whichever column the user selected |

## Change Leaderboard Filter Method:

| Actor Action | System Response |
|---|---|
| 1. Clicks "Change Filter" | 2. Displays list of filters the user can use on the leaderboard |
| 3. Selects the filter they wish to view the leaderboard under | 4. Displays the filtered Leaderboard |

## Chat with clan

| Actor Action | System Response |
|---|---|
| 1. Selects "Clan Chat" | 2. Clan chat screen displayed |

## Join Game Lobby

| Actor Action | System Response |
|---|---|

| | |
|---|---|
| 1. Selects "Join Game" | 2. Enters user into matchmaking. User is put into a suitable lobby. Starts countdown to next game. |
| 3. Can edit avatar while waiting in lobby | 4. Game is launched once countdown reaches 0 |
| Alternative flow:<br>2a. Lobby user is entered into is already in a game<br>• User joins game without a countdown | |

Non Functional Requirement: Usability

The UI for joining a game should be user friendly and easy to use.

## Leave Game Lobby

| Actor Action | System Response |
|---|---|
| 1. Selects "Leave Lobby / Game" | 2. User is returned to the home screen. Users profile is updated to no longer being in a session. |

## Create Private Session

| Actor Action | System Response |
|---|---|
| 1. Selects "Create Private Session" | 2. Displays Private Session Creation Screen |
| 3. Choses game mode, arena, game rules. | 4. Sets up lobby based on users specifications |
| 5. Invites players from clan to join | 6. Launches game once all players ready up |

## View Clan Calendar

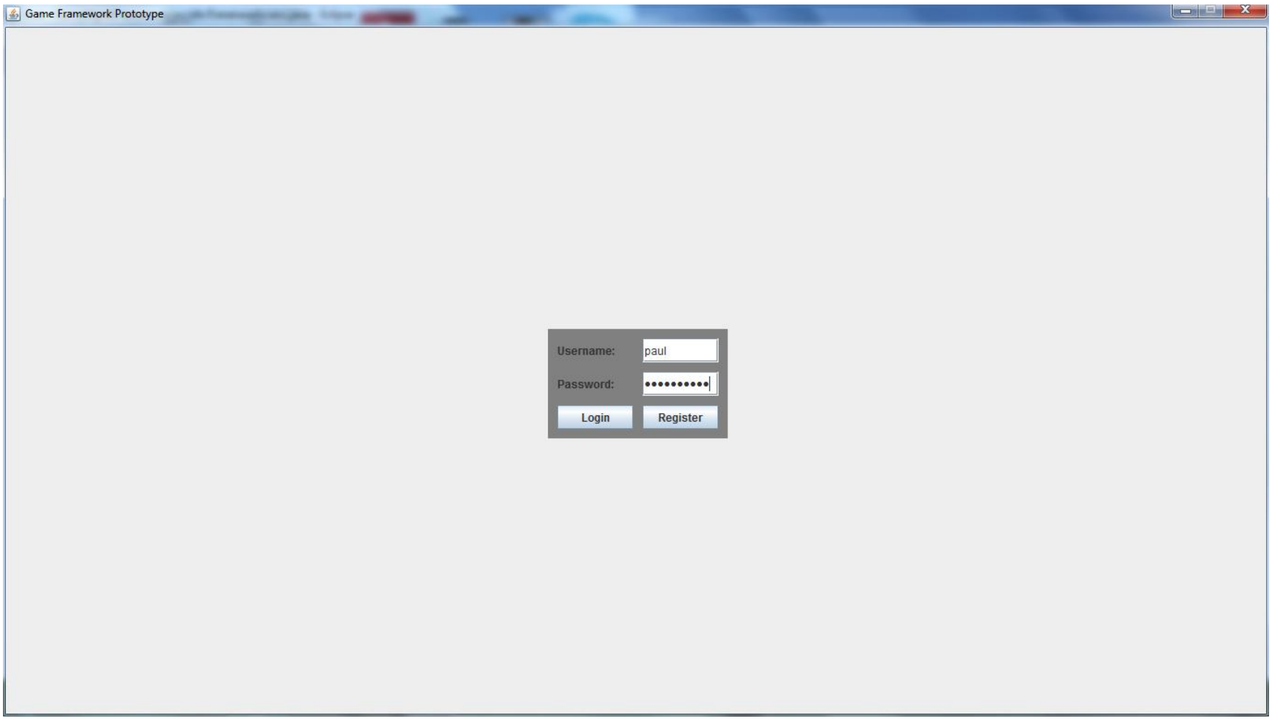
| Actor Action | System Response |
|---|---|
| 1. Selects "View Calendar" | 2. Displays calendar with upcoming events marked on relevant dates. |

## Visit Forums

| Actor Action | System Response |
|---|---|
| 1. Clicks "Visit Clan Forum" | 2. Launches user's internet browser and opens the forums website. |

## Screen Shots

## Login Window

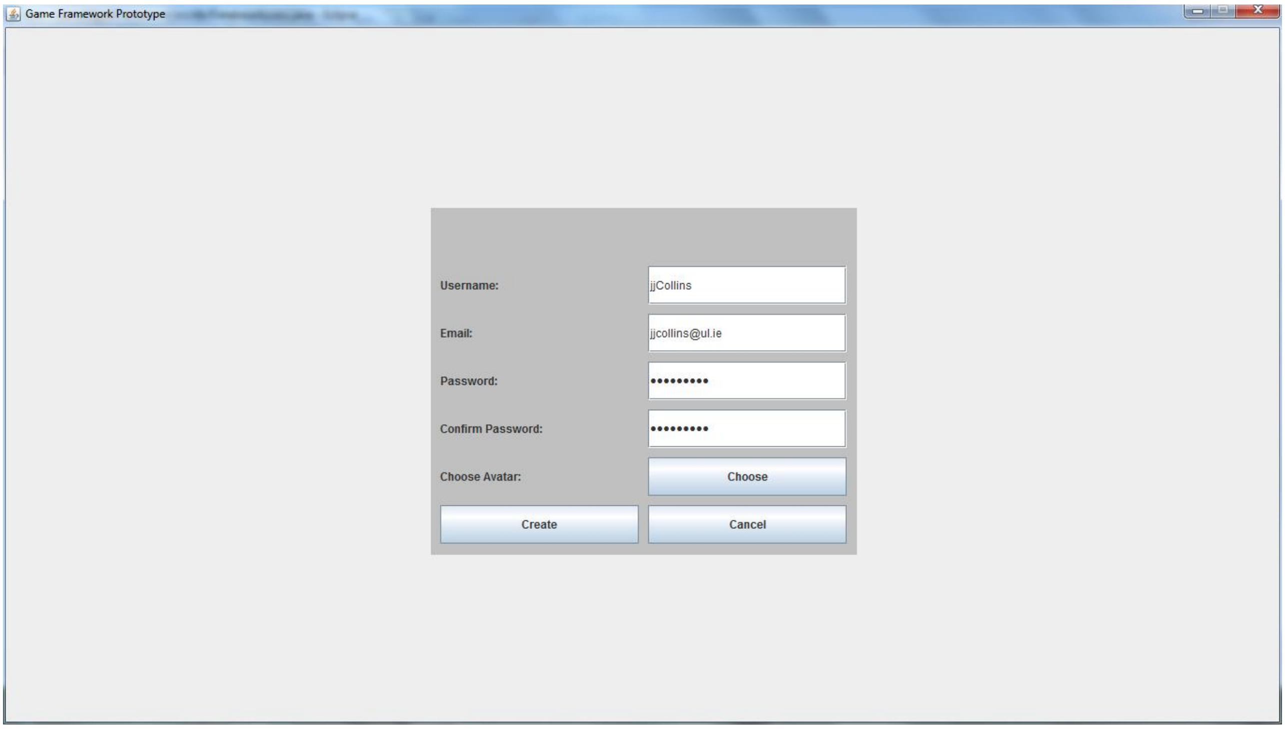


Game Framework Prototype

Username: paul

Password: ••••••••••

## User Registration



Game Framework Prototype

Username: jjCollins

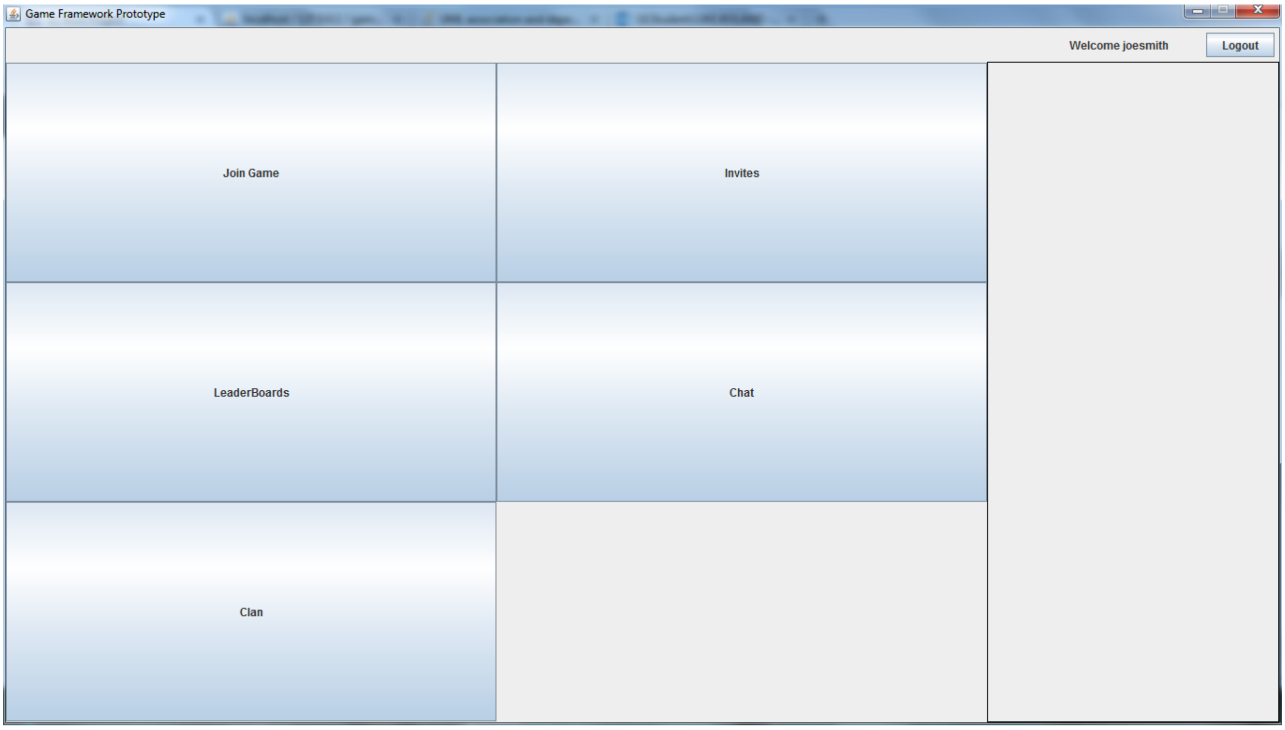
Email: jjcollins@ul.ie

Password: •••••••••

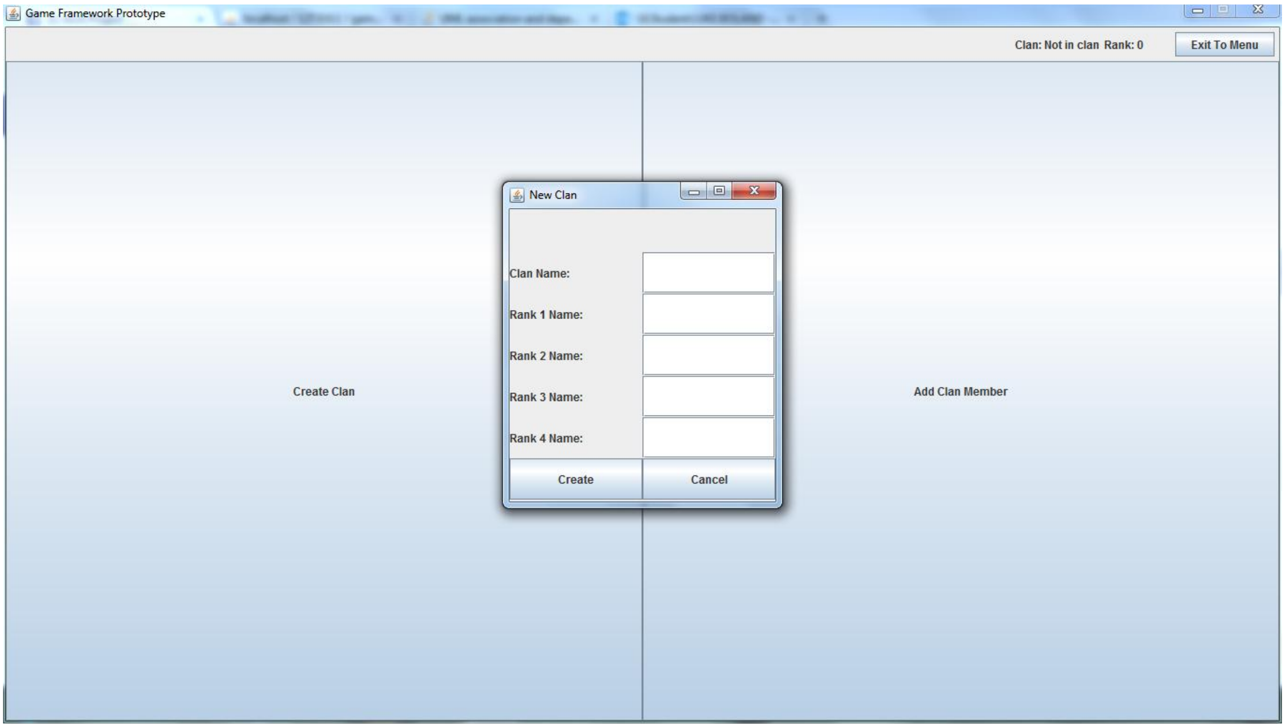
Confirm Password: •••••••••

Choose Avatar:

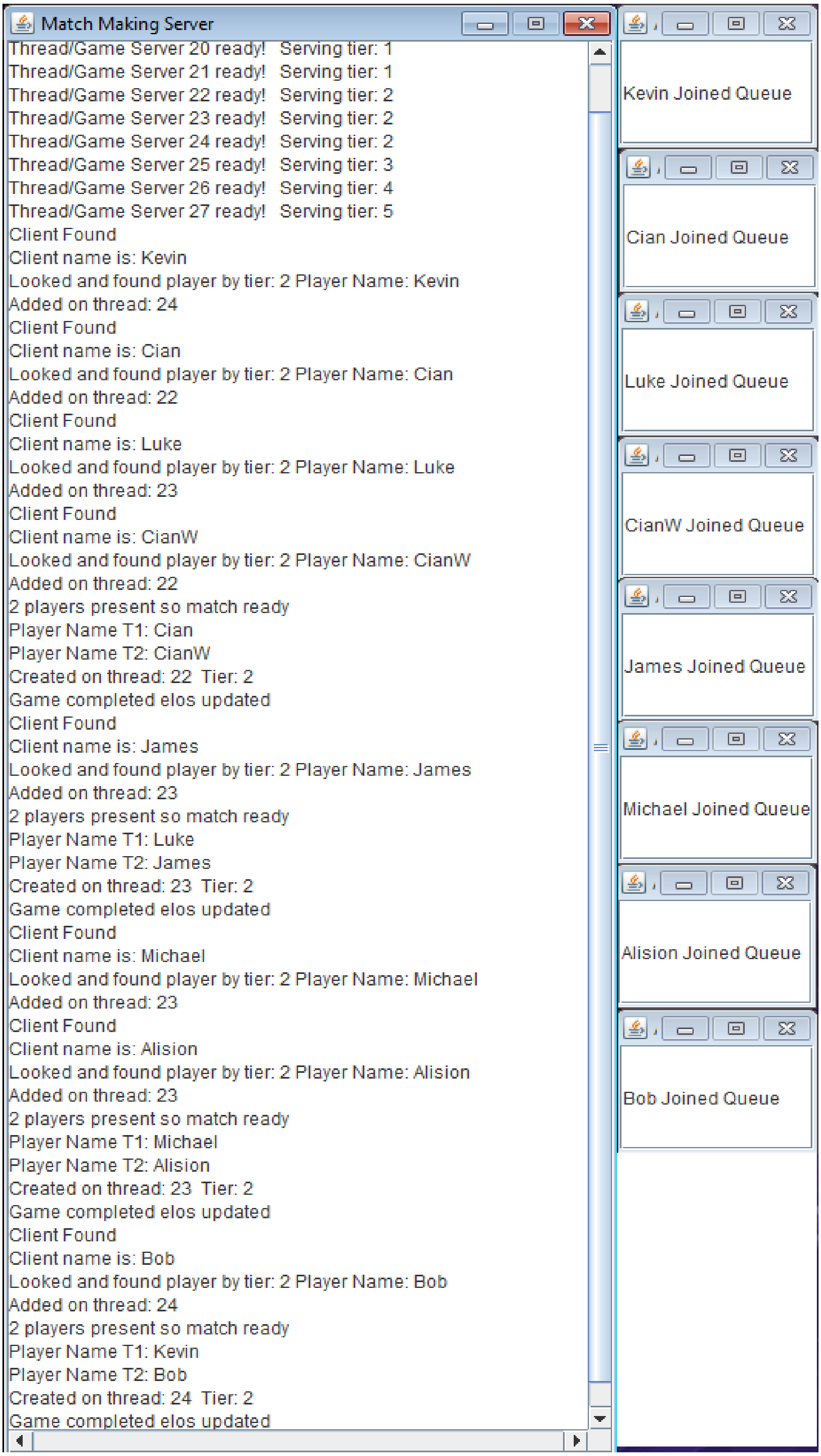
## Home Screen



## Clan Creation



## Match Making



Match Making Server

```
Thread/Game Server 20 ready!   Serving tier: 1
Thread/Game Server 21 ready!   Serving tier: 1
Thread/Game Server 22 ready!   Serving tier: 2
Thread/Game Server 23 ready!   Serving tier: 2
Thread/Game Server 24 ready!   Serving tier: 2
Thread/Game Server 25 ready!   Serving tier: 3
Thread/Game Server 26 ready!   Serving tier: 4
Thread/Game Server 27 ready!   Serving tier: 5
Client Found
Client name is: Kevin
Looked and found player by tier: 2 Player Name: Kevin
Added on thread: 24
Client Found
Client name is: Cian
Looked and found player by tier: 2 Player Name: Cian
Added on thread: 22
Client Found
Client name is: Luke
Looked and found player by tier: 2 Player Name: Luke
Added on thread: 23
Client Found
Client name is: CianW
Looked and found player by tier: 2 Player Name: CianW
Added on thread: 22
2 players present so match ready
Player Name T1: Cian
Player Name T2: CianW
Created on thread: 22  Tier: 2
Game completed elos updated
Client Found
Client name is: James
Looked and found player by tier: 2 Player Name: James
Added on thread: 23
2 players present so match ready
Player Name T1: Luke
Player Name T2: James
Created on thread: 23  Tier: 2
Game completed elos updated
Client Found
Client name is: Michael
Looked and found player by tier: 2 Player Name: Michael
Added on thread: 23
Client Found
Client name is: Alision
Looked and found player by tier: 2 Player Name: Alision
Added on thread: 23
2 players present so match ready
Player Name T1: Michael
Player Name T2: Alision
Created on thread: 23  Tier: 2
Game completed elos updated
Client Found
Client name is: Bob
Looked and found player by tier: 2 Player Name: Bob
Added on thread: 24
2 players present so match ready
Player Name T1: Kevin
Player Name T2: Bob
Created on thread: 24  Tier: 2
Game completed elos updated
```

```
Kevin Joined Queue
Cian Joined Queue
Luke Joined Queue
CianW Joined Queue
James Joined Queue
Michael Joined Queue
Alision Joined Queue
Bob Joined Queue
```

# Analysis Artefacts

## Data Driven Design

### Candidate Objects

We used the 'noun identification' technique in order to list any possible object. After building the initial list we then filtered it using a set of basic heuristics in order to specify more meaningful objects.

The initial list constructed (without duplicates):

| window | user | form | game | dashboard |
|---|---|---|---|---|
| system | information | click | session | clan |
| server | forum | address | actor | manager |
| Event | screen | member | invite | text |
| Email | profile | message | rank | menu |
| Option | List | display | name | chat |
| calendar | Date | participant | leader board | view |
| column | Filter | lobby | avatar | match maker |
| Timer | Home | mode | arena | player |
| browser | Website | | | |

Heuristics applied:

1. Too vague or specific - RED.
2. Out of scope - BLUE.
3. Attribute - GREEN.
4. Operation - YELLOW.
5. Equivalent (other similar objects) - PURPLE.

The filtered list:

| Server | clan event(event) | calendar | timer |
|---|---|---|---|
| Player | match maker | screen | profile |
| Message | Lobby | invite | leader board |
| Dashboard | manager | clan | |

*Server*

The framework is intended to be designed on a distributed system of servers. At the most basic level of that system, there are three relevant servers, but only two that are integral to the framework:

1. The match making server.
2. The database server.
3. The game server.

While the game server interacts with the framework, it is out of the scope of our system. The system is focused on how to distribute players into teams and games fairly, finally passing these distributions onto the game server which hosts the game. The matchmaking server and database server are independent of each other and the client. We can interact with them through a network manager.

The database accepts requests to pull data from the server for clients. It also allows data to be updated both by the client and matchmaker.

### *Player*

A player is the user of the client application. A player is an entity within a game and as such contains relevant information like kills/deaths/wins/etc. A clan is made up of, and headed by a number of players. Players are organised into teams by the match maker.

### *Message*

A message is data given from one player to another through text. A message contains the text and the ids of the player sender and player recipient.

### *Clan Event (Event)*

The framework has its own definition of an event. An event can be created by a Clan leader and an invitation to the event is sent to Clan members. To stop ambiguity between a system event and a clan event the event object will be renamed to "ClanEvent".

### *Matchmaker*

The matchmaker accepts player objects and divides them into appropriate teams. In a fully developed, distributed system, there would be several instances of this server waiting for clients to connect. The server, after distributing players, would then pass the teams over to the game servers.

### *Lobby*

The lobby is a section of the Join Game process (use case) that allows players to view their teammates and customise their ‘avatar’ before joining a game.

### *Manager*

A generic term that encompasses the classes that delegate data, communicate between servers, etc. Talked about in more detail in the Responsibility Driven Design section.

## Responsibility Driven Design

The noun verification technique did not answer all our questions. How was our data going to be managed? Who (i.e. what class) would delegate data to the servers, where are the IP addresses listed, etc.

While our priority was not to create a distributed system, we still needed a proof of concept of a class that could handle talking to different types of servers, establishing connections, and relaying data.

### The Network Manager

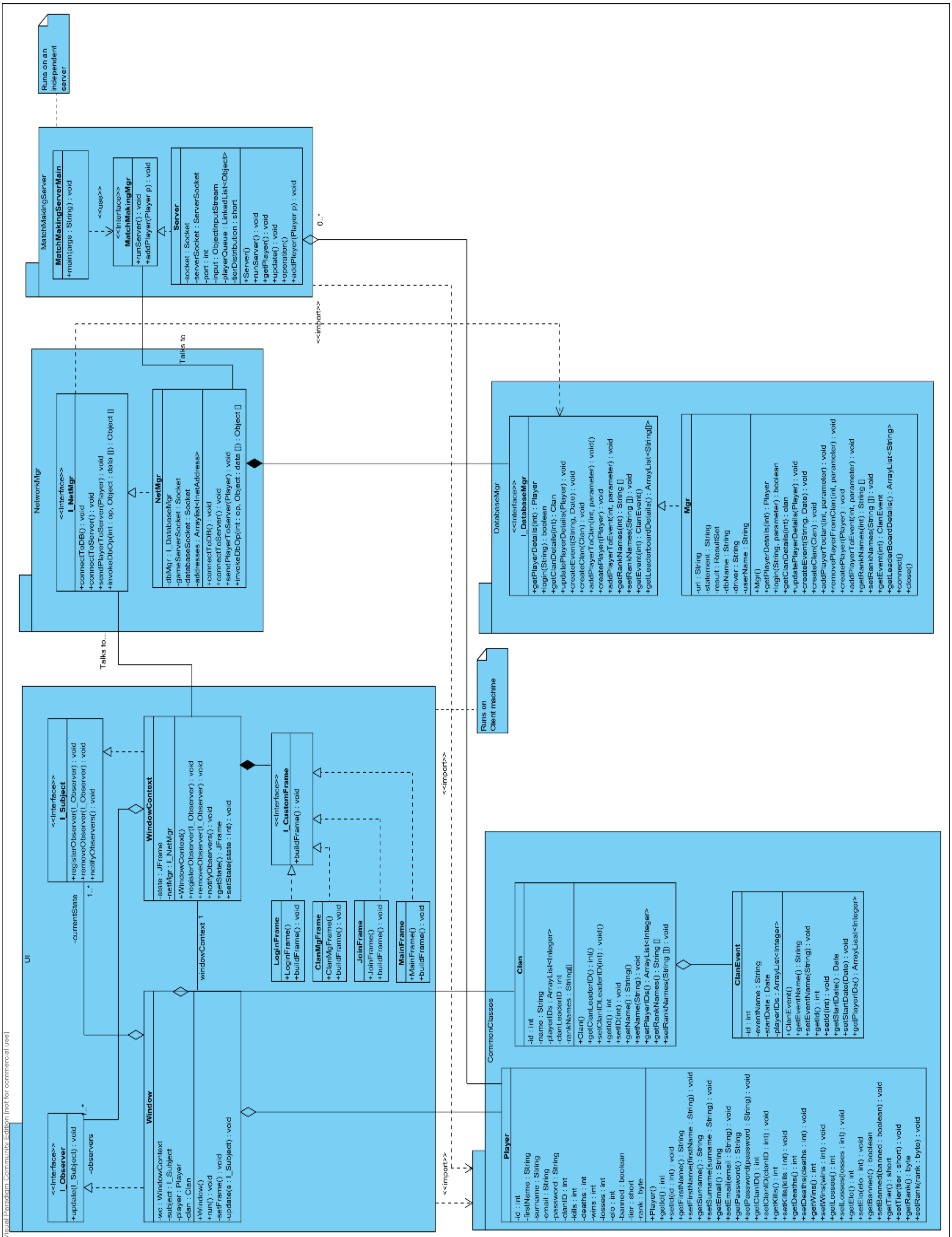
The network manager is responsible for:

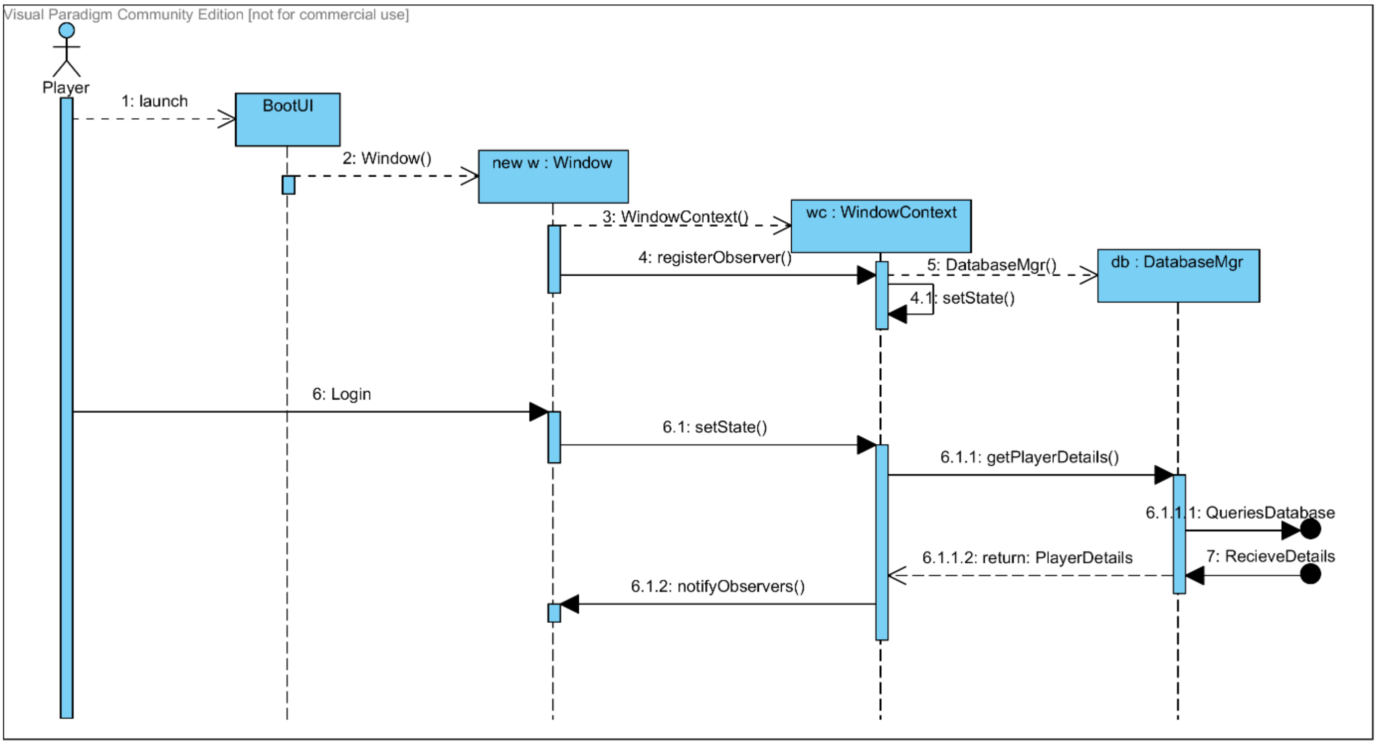
1. Establishing a connection to the match maker.
2. Establishing a connection to the database server.
3. (Post-Match Making) Receive a connection to the game server.
4. Creating a heartbeat monitor to check if a server is alive.
5. Managing connection issues/failures in a graceful manner (no client side crashes).

## Class Diagram

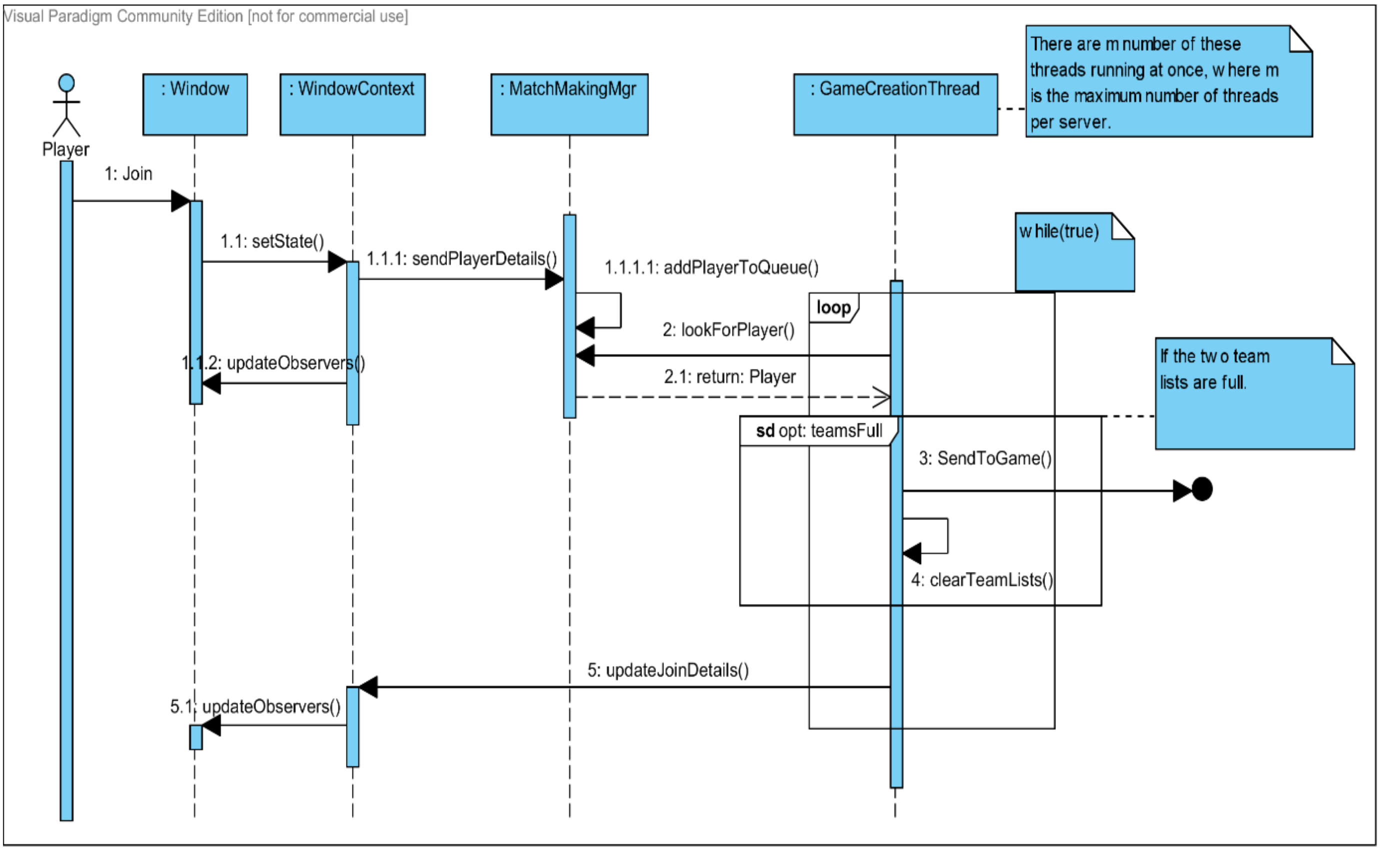
## Interaction Diagram

### Login


Runs on an independent server
MatchMakingServer
MatchMakingServerMain
+main(args : String) : void
<<use>>
<<Interface>>
MatchMakingMgr
+runServer() : void
+addPlayer(Player p) : void
Server
-socket : Socket
-serverSocket : ServerSocket
-port : int
-input : ObjectInputStream
-playerQueue : LinkedList<Object>
-tierDistribution : short
+Server()
+runServer() : void
+getPlayer() : void
+update() : void
+operation()
+addPlayer(Player p) : void
0..*
<<import>>
Talks to
NetworkMgr
<<Interface>>
I_NetMgr
+connectToDB() : void
+connectToServer() : void
+sendPlayerToServer(Player) : void
+invokeDbOp(int : op, Object : data []) : Object []
NetMgr
-dbMgr : I_DatabaseMgr
-gameServerSocket : Socket
-databaseSocket : Socket
-addresses : ArrayList<InetAddress>
+connectToDB() : void
+connectToServer() : void
+sendPlayerToServer(Player) : void
+invokeDbOp(int : op, Object : data []) : Object []
DatabaseMgr
<<Interface>>
I_DatabaseMgr
+getPlayerDetails(int) : Player
+login(String) : boolean
+getClanDetails(int) : Clan
+updatePlayerDetails(Player) : void
+createEvent(String, Date) : void
+createClan(Clan) : void
+addPlayerToClan(int, parameter) : void()
+createPlayer(Player) : void
+addPlayerToEvent(int, parameter) : void
+getRankNames(int) : String []
+setRankNames(String []) : void
+getEvent(int) : ClanEvent()
+getLeaderboardDetails() : ArrayList<String[]>
Mgr
-url : String
-statement : String
-result : ResultSet
-dbName : String
-driver : String
-userName : String
+Mgr()
+getPlayerDetails(int) : Player
+login(String, parameter) : boolean
+getClanDetails(int) : clan
+updatePlayerDetails(Player) : void
+createEvent(String, Date) : void
+createClan(Clan) : void
+addPlayerToclan(int, parameter) : void
+removePlayerFromClan(int, parameter) : void
+createPlayer(Player) : void
+addPlayerToEvent(int, parameter) : void
+getRankNames(int) : String []
+setRankNames(String []) : void
+getEvent(int) : ClanEvent
+getLeaderBoardDetails() : ArrayList<String>
+connect()
+close()
Talks to...
Runs on Client machine
<<import>>
UI
<<Interface>>
I_Subject
+registerObserver(I_Observer) : void
+removeObserver(I_Observer) : void
+notifyObservers() : void
WindowContext
-state : JFrame
-netMgr : I_NetMgr
+WindowContext()
+registerObserver(I_Observer) : void
+removeObserver(I_Observer) : void
+notifyObservers() : void
+getState() : JFrame
+setState(state : int) : void
<<Interface>>
I_CustomFrame
+buildFrame() : void
LoginFrame
+LoginFrame()
+buildFrame() : void
ClanMgFrame
+ClanMgFrame()
+buildFrame() : void
JoinFrame
+JoinFrame()
+buildFrame() : void
MainFrame
+MainFrame()
+buildFrame() : void
-currentState
1..*
windowContext 1
-observers
1..*
<<Interface>>
I_Observer
+update(I_Subject) : void
Window
-wc : WindowContext
-subject : I_Subject
-player : Player
-clan : Clan
+Window()
+run() : void
-setFrame() : void
-update(s : I_Subject) : void
<<import>>
CommonClasses
Clan
-id : int
-name : String
-playerIDs : ArrayList<Integer>
-clanLeaderID : int
-rankNames : String[]
+Clan()
+getClanLeaderID() : int()
+setClanLeaderID(int) : void()
+getId() : int
+setId(int) : void
+getName() : String
+setName(String) : void
+getPlayerIDs() : ArrayList<Integer>
+getRankNames() : String []
+setRankNames(String []) : void
ClanEvent
-id : int
-eventName : String
-startDate : Date
-playerIDs : ArrayList<Integer>
+ClanEvent()
+getEventName() : String
+setEventName(String) : void
+getId() : int
+setId(int) : void
+getStartDate() : Date
+setStartDate(Date) : void
+getPlayerIDs() : ArrayList<Integer>
Player
-id : int
-firstName : String
-surname : String
-email : String
-password : String
-clanID : int
-kills : int
-deaths : int
-wins : int
-losses : int
-elo : int
-banned : boolean
-tier : short
-rank : byte
+Player()
+getId() : int
+setId(id : int) : void
+getFirstName() : String
+setFirstName(firstName : String) : void
+getSurname() : String
+setSurname(surname : String) : void
+getEmail() : String
+setEmail(email : String) : void
+getPassword() : String
+setPassword(password : String) : void
+getClanID() : int
+setClanID(clanID : int) : void
+getKills() : int
+setKills(kills : int) : void
+getDeaths() : int
+setDeaths(deaths : int) : void
+getWins() : int
+setWins(wins : int) : void
+getLosses() : int
+setLosses(losses : int) : void
+getElo() : int
+setElo(elo : int) : void
+getBanned() : boolean
+setBanned(banned : boolean) : void
+getTier() : short
+setTier(tier : short) : void
+getRank() : byte
+setRank(rank : byte) : void
Visual Paradigm Community Edition [not for commercial use]


Visual Paradigm Community Edition [not for commercial use]
Player
1: launch
BootUI
2: Window()
new w : Window
3: WindowContext()
wc : WindowContext
4: registerObserver()
5: DatabaseMgr()
db : DatabaseMgr
4.1: setState()
6: Login
6.1: setState()
6.1.1: getPlayerDetails()
6.1.1.1: QueriesDatabase
7: RecieveDetails
6.1.1.2: return: PlayerDetails
6.1.2: notifyObservers()

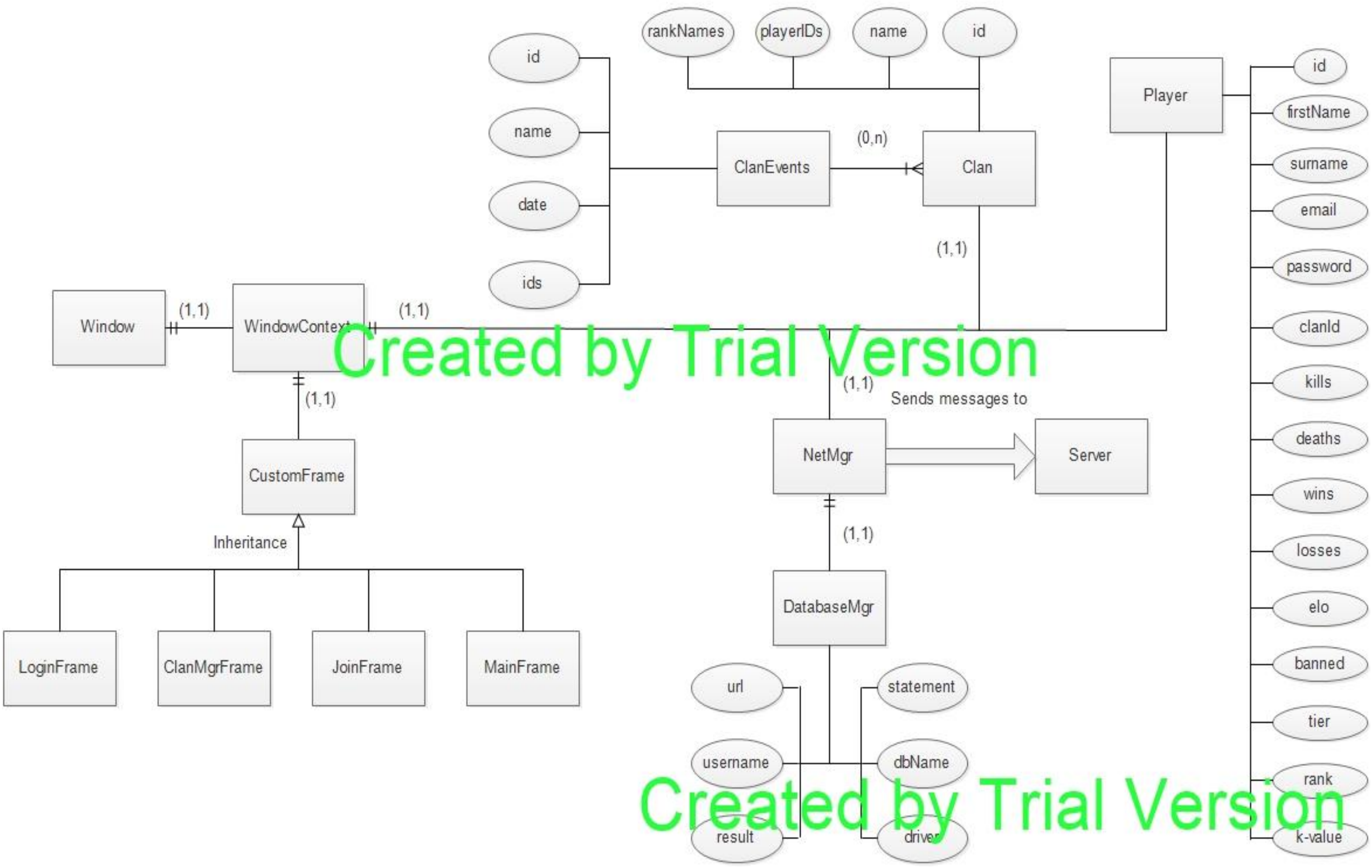
## Join Game


Visual Paradigm Community Edition [not for commercial use]
Player
: Window
: WindowContext
: MatchMakingMgr
: GameCreationThread
There are m number of these threads running at once, w here m is the maximum number of threads per server.
1: Join
1.1: setState()
1.1.1: sendPlayerDetails()
1.1.1.1: addPlayerToQueue()
w hile(true)
loop
2: lookForPlayer()
1.1.2: updateObservers()
2.1: return: Player
If the tw o team lists are full.
sd opt: teamsFull
3: SendToGame()
4: clearTeamLists()
5: updateJoinDetails()
5.1: updateObservers()

## E-R Diagram

Created by Trial Version

Entity - Relationship Diagram for Games Matchmaking Framework



Created by Trial Version

Created by Trial Version

# System Architecture/Design

## Discussion

### Architectural Decisions Taken

The system for this project is divided into three separate sub-systems. The matchmaking service and the data service will each run independently on their own servers with the third system being the client system running on the user's computer.  In the client system we chose to use the MVC architectural pattern separating the system into the Panels / Window classes as the view, the ProcessInput class as the controller and the Player / Database Interface Classes. This separates the user interface from the business logic allowing for the system to be more maintainable. This also allows views to frequently change in design with minimal changes to the model and controller unless perhaps more features are being added. The factory pattern can be seen with the Panel Interface in Panel.java which is realised by BootUI.java, ClanUI.java, HomeUI.java, JoinGameUI.java, LeaderboardUI.java and MenuUI.java with the factory implemented in PanelFactory.java

### UML Workbench

We chose Visual Paradigm as our UML workbench. Initially we sampled a number of programs that were available to us at no cost namely Gliffy, EDraw and Creately. However, we found each had their own faults.

Gliffy and Creately are online based workbenches with limited functions. EDraw had massive UML support but we found that the diagrams weren't as sharp and as professional looking as Visual Paradigm. We also found that the program ran quite slow taking longer than would be expected to perform operations in comparison to Visual Paradigm.

Visual Paradigm extensively supports UML 2 in its free community edition and also gave us limited access to code generation tools from diagrams to gain more familiarity on the concept.

### Patterns Description

A number of design patterns are used in the three systems. In the client system the user interface windowing is implemented using a combination of the observer and factory pattern. The factory pattern is a creational design pattern that abstracts the instantiation logic from the client and refers to the newly created object through a common interface. In our case it allows us to create multiple types of panel that can all be accessed through the one panel interface. This makes it easier to develop new panels for the system when new user interface features are being added, making the system more extensible and maintainable for future versions.

That combined with the observer design pattern, which is a behavioural design pattern that defines a one to one dependency between objects so that when an object changes state all of its dependants are notified and automatically updated, allows us to take any panel instantiated by the panel factory and automatically update the window with this panel as the window is subscribed to that panel interface. The observer pattern allows many objects to subscribe to the subject, in our case we only ever want one window present at any time. The observer interfaces can be seen in Observer.java and Subject.java which contain the interfaces for the observer pattern. These are realised in PanelManager.java and Window.java.

To overcome this we used the singleton design pattern on the window object. The singleton pattern which is a creational pattern ensures that only one instance of a class is ever created and provides a global point of access to the object. This ensures only one window will ever be instantiated. The singleton pattern is also used to implement the Player class that encapsulates all the data related to the user. Only one instance of the player is needed on the client system as only one person can be logged into a client system at any given time. The singleton pattern can be seen in Player.java, ProcessInput.java and Window.java.

## Implementation Language

We chose the Java programming language to implement our client system to make the system more portable. The system can then be available to any architecture that supports the java virtual machine (JVM). Due to time constraints and also for the sake of portability among team members for the project we also wrote the database system and the matchmaking service in java. If we were to fully develop these systems we would have chosen to develop them using the C++ programming language.

For server side systems where portability is not a requirement C++ has a massive advantage over Java in terms of speed. Java has the overhead of running a virtual machine to process its bytecode into code that will run on the specific architecture below the virtual machine, hence Java’s advantage of portability as the bytecode can be taken from one machine and run on any other with the JVM.

Java also has the overhead of added checks during runtime that for the sake of speed that are not in C++. One such check for example is to check an index out of bounds in an array. Java will throw the error whereas C++ won’t as it requires an extra check for each index in an array. Speed is a major concern on any of our server side systems as the number of users that could be accessing the system at any time could potentially be massive so cutting down on processing time as much as possible is key.

## Heartbeat Monitor

A heartbeat monitoring system monitors your systems process at defined intervals set based on the priority of keeping the process alive. With single server based architecture there is very few points of failure, so when a system goes down it is not as troublesome to detect and fix the fault as opposed to a multi-server based architecture going down.

Manually monitoring a multi-server based architecture is virtually impossible without a heartbeat monitoring service in your system. Some processes have a heartbeat that might ping the monitoring service once a day if their contribution to the system is not necessary for the system to stay running. An example of this might be a process that schedules emails to be sent.

Other processes considered critical to the system could be pinged every few minutes to ensure the system is still running and that a fault can be detected before there may be a complete loss of service to the client of the system.

Heartbeats could also have a two stage warning system where a warning timeout could be triggered if a ping isn’t returned and at a later stage if another ping isn’t returned a critical timeout could be triggered. Other more critical processes could trigger a critical timeout first if the problem could be an urgent matter.

Other benefits of the a heartbeat monitoring system are it could serve as a reminder system for doing tasks that are solely manual and having a monitor on third party components that might go down with fault being on the third party developers part.

A heartbeat monitor would be beneficial to our system as our system is distributed over many servers each crucial to keeping the service running. Our database on its own system of servers is crucial to providing our client and our matchmaking servers with data to allow the user to interact with the user interface and allow the matchmaking service access to data that is crucial to its algorithms to placing users in games.

The matchmaking service will also be distributed over multiple servers to handle high amounts of users to be placed into games at any given time. The heart beat monitor would ensure the service is kept running by notifying maintenance personnel of the issue as soon as it arises before it causes major outage of service.

### Database Traffic Management Solution

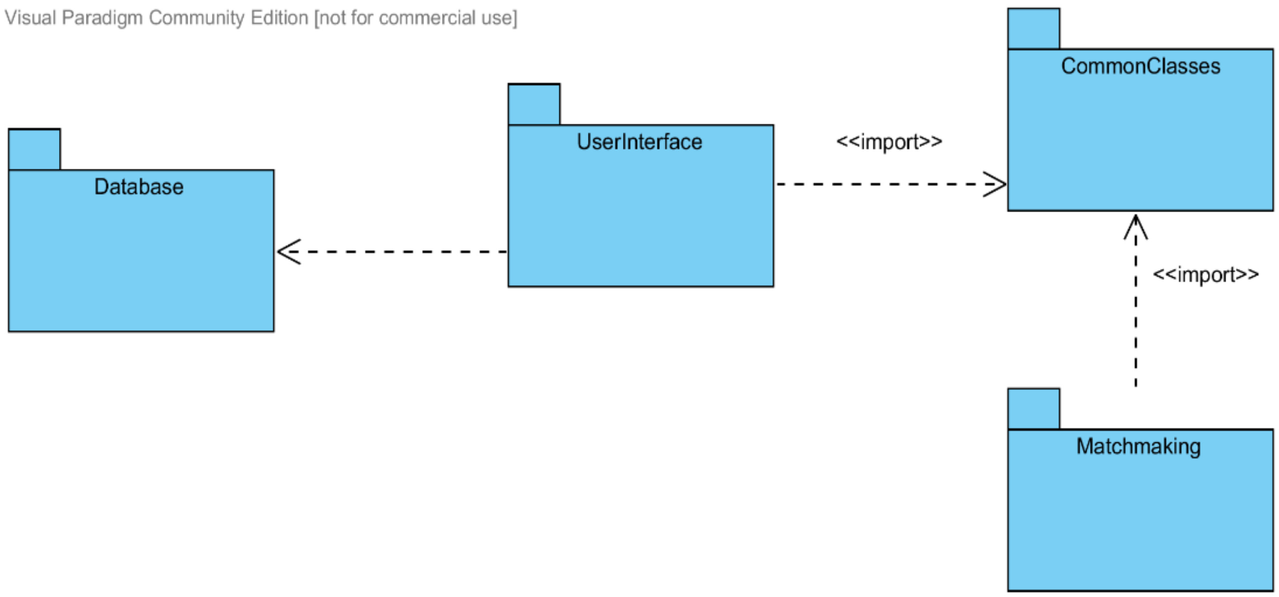
To minimise traffic on the database server we came up with a caching style system but was not implemented in our prototype. In this system the client system would locally store all information it needed about the user to provide full functionality to the client system and matchmaking service.

The data would also contain a timestamp of when the data was last pulled from the database. The data on the database would also have a timestamp of when data was last written to the database. When the user logs into the client system or is being sent into the match making service the client system queries the database for the timestamp to compare to the timestamp in local storage on the client system.

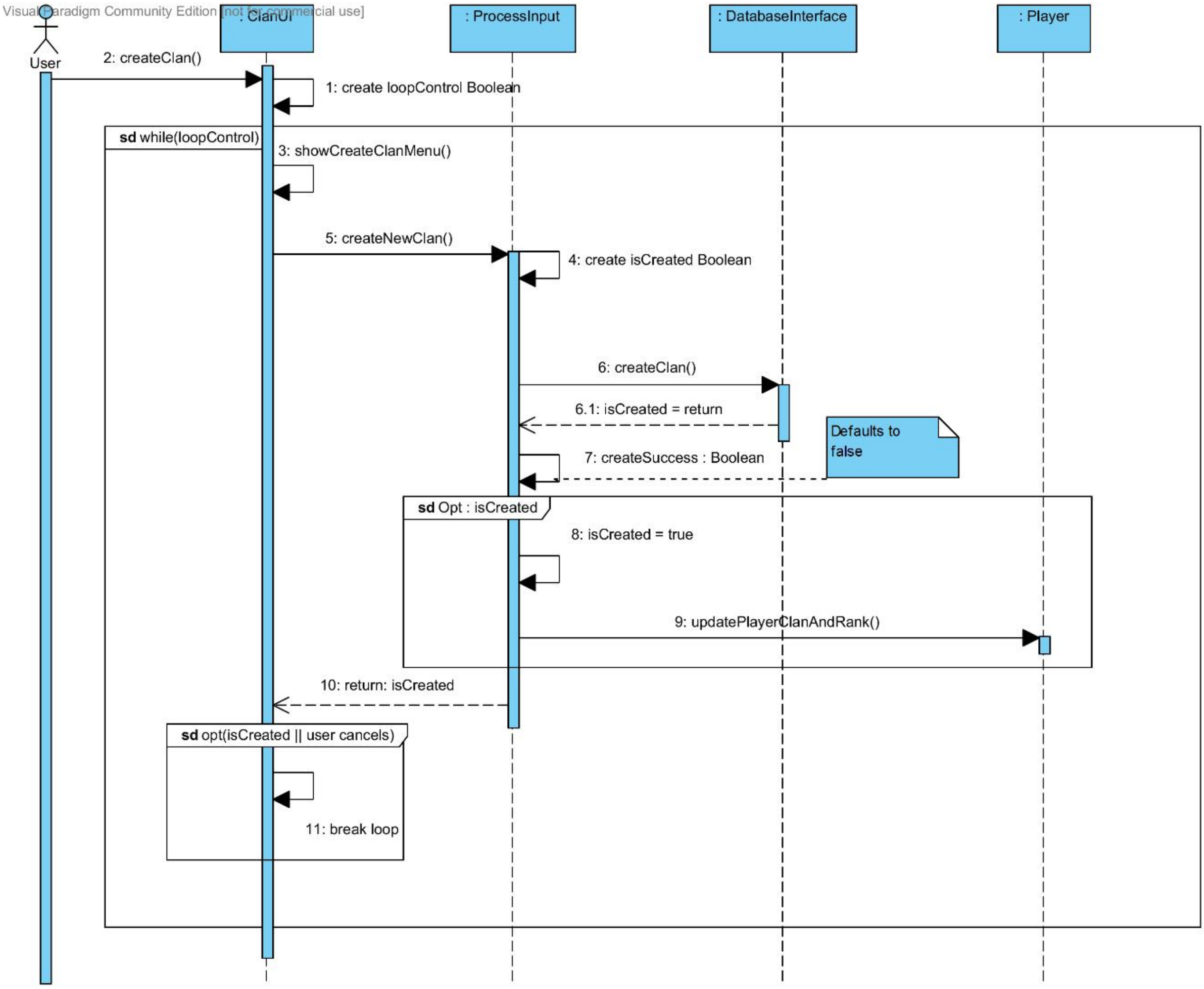
If the timestamp matched the timestamp on the database no data had been updated and the client system can use its locally stored data instead of querying the database for fresh data. This minimises traffic on the database server as a timestamp is only being requested and sent from the database rather than multiple queries requesting larger sizes of data.

## Diagrams

### Architectural Diagram

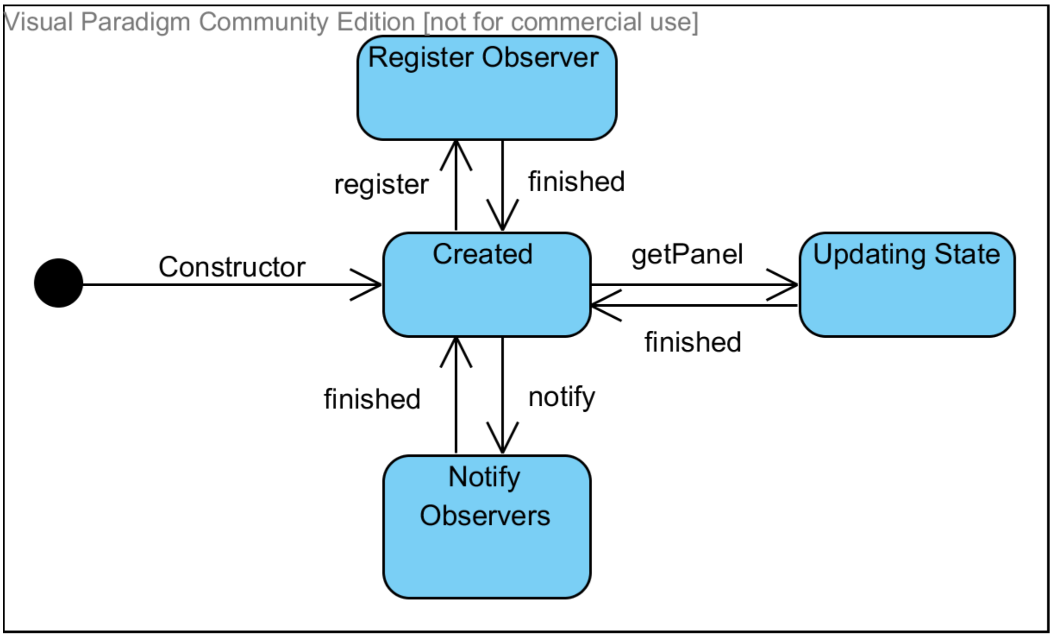


## Sequence Diagram

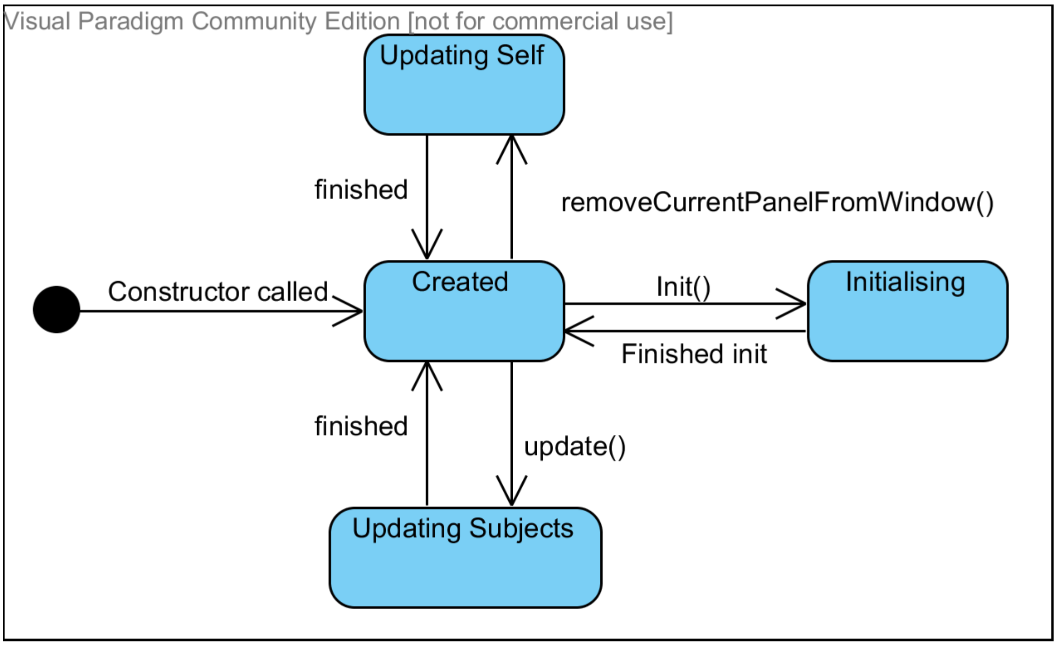


## State Diagram

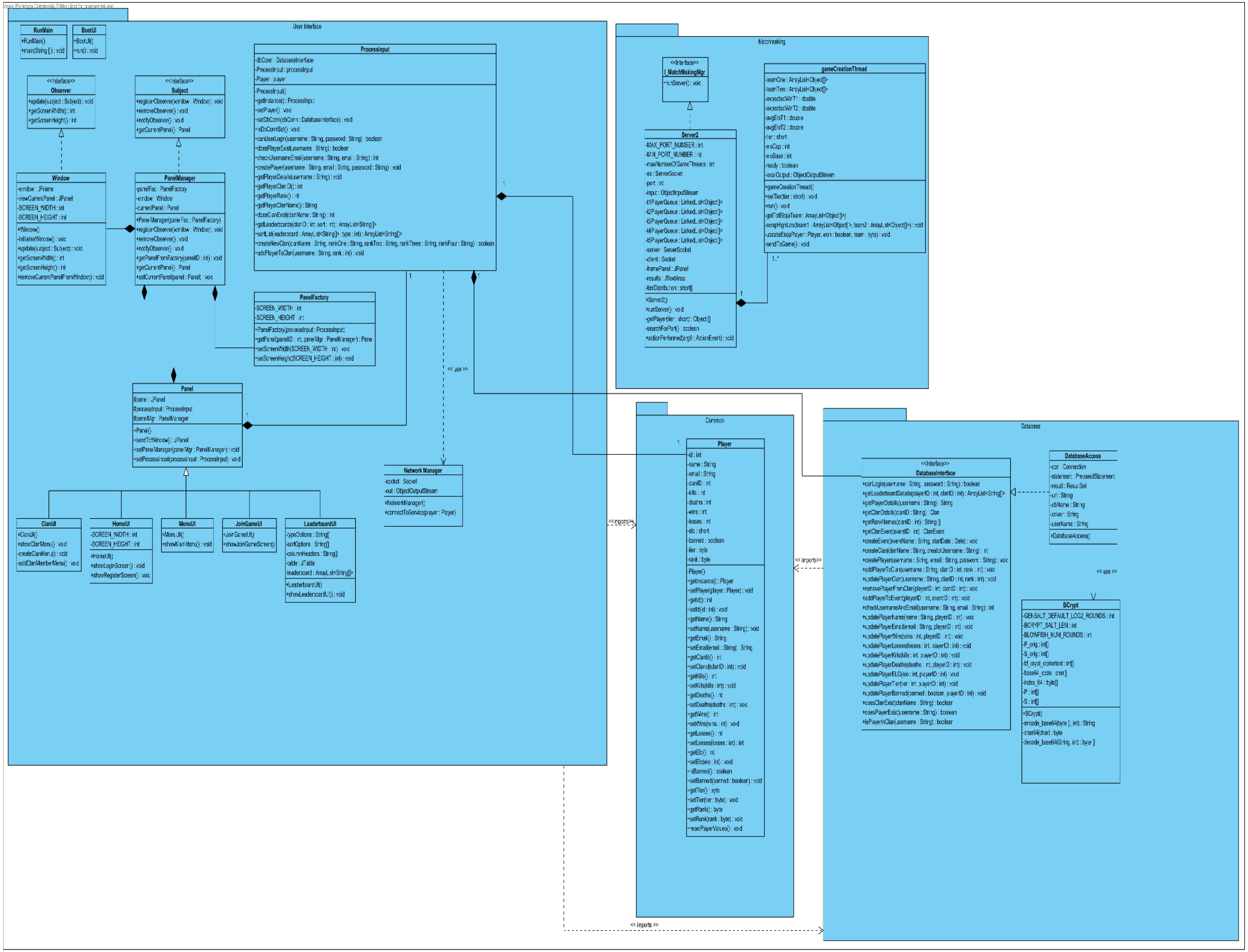
### *Panel Manager*



### *Window*



## Class Diagram


Visual Paradigm Community Edition [not for commercial use]
User Interface
RunMain
+RunMain()
+main(String [] : void
BootUI
+BootUI()
+run() : void
<<Interface>>
Observer
+update(subject : Subject) : void
+getScreenWidth() : int
+getScreenHeight() : int
<<Interface>>
Subject
+registerObserver(window : Window) : void
+removeObserver() : void
+notifyObserver() : void
+getCurrentPanel() : Panel
Window
-window : JFrame
-viewCurrentPanel : JPanel
-SCREEN_WIDTH : int
-SCREEN_HEIGHT : int
+Window()
-initialiseWindow() : void
+update(subject : Subject) : void
+getScreenWidth() : int
+getScreenHeight() : int
+removeCurrentPanelFromWindow() : void
PanelManager
-panelFac : PanelFactory
-window : Window
-currentPanel : Panel
+PanelManager(panelFac : PanelFactory)
+registerObserver(window : Window) : void
+removeObserver() : void
+notifyObserver() : void
+getPanelFromFactory(panelID : int) : void
+getCurrentPanel() : Panel
+setCurrentPanel(panel : Panel) : void
ProcessInput
-dbConn : DatabaseInterface
-ProcessInput : processInput
-Player : player
-ProcessInput()
+getInstance() : ProcessInput
+setPlayer() : void
+setDbConn(dbConn : DatabaseInterface) : void
+isDbConnSet() : void
+canUserLogin(username : String, password : String) : boolean
+doesPlayerExist(username : String) : boolean
+checkUsernameEmail(username : String, email : String) : int
+createPlayer(username : String, email : String, password : String) : void
+getPlayerDetails(username : String) : void
+getPlayerClanID() : int
+getPlayerRank() : int
+getPlayerClanName() : String
+doesClanExist(clanName : String) : int
+getLeaderboards(clanID : int, sort : int) : ArrayList<String[]>
+sortList(leaderboard : ArrayList<String[]>, type : int) : ArrayList<String[]>
+createNewClan(clanName : String, rankOne : String, rankTwo : String, rankThree : String, rankFour : String) : boolean
+addPlayerToClan(username : String, rank : int) : void
PanelFactory
-SCREEN_WIDTH : int
-SCREEN_HEIGHT : int
+PanelFactory(processInput : ProcessInput)
+getPanel(panelID : int, panelMgr : PanelManager) : Pane
+setScreenWidth(SCREEN_WIDTH : int) : void
+setScreenHeight(SCREEN_HEIGHT : int) : void
Panel
#panel : JPanel
#processInput : ProcessInput
#panelMgr : PanelManager
+Panel()
+sendToWindow() : JPanel
+setPanelManager(panelMgr : PanelManager) : void
+setProcessInput(processInput : ProcessInput) : void
Network Manager
-socket : Socket
-out : ObjectOutputStream
+NetworkManager()
+connectToService(player : Player)
ClanUI
+ClanUI()
+showClanMenu() : void
-createClanMenu() : void
-addClanMemberMenu() : void
HomeUI
-SCREEN_WIDTH : int
-SCREEN_HEIGHT : int
+HomeUI()
+showLoginScreen() : void
+showRegisterScreen() : void
MenuUI
+MenuUI()
+showMainMenu() : void
JoinGameUI
+JoinGameUI()
+showJoinGameScreen()
LeaderboardUI
-typeOptions : String[
-sortOptions : String[]
-columnHeaders : String[]
-table : JTable
-leaderboard : ArrayList<String[]>
+LeaderboardUI()
+showLeaderboardUI() : void
1
1
1
1
<< use >>
Matchmaking
<<Interface>>
I_MatchMakingMgr
+runServer() : void
Server2
-MAX_PORT_NUMBER : int
-MIN_PORT_NUMBER : int
-maxNumberOfGameThreads : int
-ss : ServerSocket
-port : int
-input : ObjectInputStream
-t1PlayerQueue : LinkedList<Object[]>
-t2PlayerQueue : LinkedList<Object[]>
-t3PlayerQueue : LinkedList<Object[]>
-t4PlayerQueue : LinkedList<Object[]>
-t5PlayerQueue : LinkedList<Object[]>
-server : ServerSocket
-client : Socket
-framePanel : JPanel
-results : JTextArea
-tierDistribution : short[]
+Server2()
+runServer() : void
-getPlayer(tier : short) : Object []
-searchForPort() : boolean
+actionPerformed(arg0 : ActionEvent) : void
gameCreationThread
-teamOne : ArrayList<Object[]>
-teamTwo : ArrayList<Object[]>
-expectedWinT1 : double
-expectedWinT2 : double
-avgEloT1 : double
-avgEloT2 : double
-tier : short
-eloCap : int
-eloBase : int
-ready : boolean
-localOutput : ObjectOutputStream
+gameCreationThread()
+setTier(tier : short) : void
+run() : void
-getTotElo(aTeam : ArrayList<Object[]>)
-swapHighLow(team1 : ArrayList<Object[]>, team2 : ArrayList<Object[]>) : void
-updateElo(aPlayer : Player, won : boolean, team : byte) : void
-sendToGame() : void
1..*
1
Common
Player
-id : int
-name : String
-email : String
-clanID : int
-kills : int
-deaths : int
-wins : int
-losses : int
-elo : short
-banned : boolean
-tier : byte
-rank : byte
-Player()
+getInstance() : Player
+setPlayer(player : Player) : void
+getId() : int
+setId(id : int) : void
+getName() : String
+setName(username : String) : void
+getEmail() : String
+setEmail(email : String) : String
+getClanId() : int
+setClanId(clanID : int) : void
+getKills() : int
+setKills(kills : int) : void
+getDeaths() : int
+setDeaths(deaths : int) : void
+getWins() : int
+setWins(wins : int) : void
+getLosses() : int
+setLosses(losses : int) : int
+getElo() : int
+setElo(elo : int) : void
+isBanned() : boolean
+setBanned(banned : boolean) : void
+getTier() : byte
+setTier(tier : byte) : void
+getRank() : byte
+setRank(rank : byte) : void
+resetPlayerValues() : void
1
<<imports>>
<< imports>>
<< imports >>
Database
<<Interface>>
DatabaseInterface
+canLogin(username : String, password : String) : boolean
+getLeaderboardDetails(playerID : int, clanID : int) : ArrayList<String[]>
+getPlayerDetails(username : String) : String
+getClanDetails(clanID : String) : Clan
+getRankNames(clanID : int) : String []
+getClanEvent(eventID : int) : ClanEvent
+createEvent(eventName : String, startDate : Date) : void
+createClan(clanName : String, creatorUsername : String) : int
+createPlayer(username : String, email : String, password : String) : void
+addPlayerToClan(username : String, clanID : int, rank : int) : void
+updatePlayerClan(username : String, clanID : int, rank : int) : void
+removePlayerFromClan(playerID : int, clanID : int) : void
+addPlayerToEvent(playerID : int, eventID : int) : void
+checkUsernameAndEmail(username : String, email : String) : int
+updatePlayerName(name : String, playerID : int) : void
+updatePlayerEmail(email : String, playerID : int) : void
+updatePlayerWins(wins : int, playerID : int) : void
+updatePlayerLosses(losses : int, playerID : int) : void
+updatePlayerKills(kills : int, playerID : int) : void
+updatePlayerDeaths(deaths : int, playerID : int) : void
+updatePlayerELO(elo : int, playerID : int) : void
+updatePlayerTier(tier : int, playerID : int) : void
+updatePlayerBanned(banned : boolean, playerID : int) : void
+doesClanExist(clanName : String) : boolean
+doesPlayerExist(username : String) : boolean
+isPlayerInClan(username : String) : boolean
DatabaseAccess
-con : Connection
-statement : PreparedStatement
-result : ResultSet
-url : String
-dbName : String
-driver : String
-userName : String
+DatabaseAccess()
<< use >>
BCrypt
-GENSALT_DEFAULT_LOG2_ROUNDS : int
-BCRYPT_SALT_LEN : int
-BLOWFISH_NUM_ROUNDS : int
-P_orig : int[]
-S_orig : int[]
-bf_crypt_ciphertext : int[]
-base64_code : char[]
-index_64 : byte[]
-P : int[]
-S : int[]
+BCrypt()
-encode_base64(byte [], int) : String
-char64(char) : byte
-decode_base64(String, int) : byte []

# Critique

## Analysis Artefacts and Design - Comparison and Contrast

### Cohesion and Coupling

*What was designed*

The noun verification technique (and the subsequent culling) introduced a variety of succinct classes. However, the initial relations between the classes, and their division between different packages, resulted in a considerable amount of coupling (particularly at the package level).

In reality we should have strived for looser coupling across the components of the system. The primary problem is the heavy reliance on the *Common Classes* package. This was initially intended as a simple solution to what appeared to be a simple problem. Since the use of the classes propagated throughout the system and no interface (or any design) was specified for these classes, we introduced high coupling along with what could eventually become unnecessary maintenance in later releases.

A common interface for the classes *Player, Clan* and *ClanEvent* are necessary due to the independent nature of the matchmaker and client. There is no direct association between the two, just passing messages. As a result, in order to maintain consistency between iterations of each client and matchmaker, interfaces should be provided.

*What should have been*

We should have more clearly defined the roles of each of the classes. For instance, since there is only one instance of the *Player* class, passing the one instantiated object to the necessary classes/components would have been more appropriate.

The *Common Classes* package itself, was poorly defined. It was essentially a storage component, which undermined the inherit usefulness of packages. Defined as an arbitrary storage unit as opposed to a component with a specific and meaningful (i.e.: more meaningful than storage) purpose like, for example, the Math class. While the *Player*, *Clan* and *ClanEvent* classes were associated (*Player* has a *Clan* which has zero or more *ClanEvent*), the system wide dependency was unnecessary.

The so called *Common Classes*, despite being so widely used, were not given an interface to adhere to, thus potentially raising the cost of maintenance by a large factor. This will be especially relevant as the system expands into a distributed system.

*What was implemented*

There were a variety of design patterns used in order to combat some of the discovered deficiencies as explained before and soon after. The *Common Classes* package was removed from the client side, now requiring only a single instance of the *Player* class and references to the *Clan* and *ClanEvent.*

### Client Observations

The original client's design incorporated an observer pattern which communicated with the matchmaking server and the data manager via the network manager. The pattern would change its state (i.e.: the frame to be displayed) to one of four custom frame objects.

Criticisms of, and updates to, the original client include:

1. The custom frames would have to be swapped/rebuilt fairly regularly, and the intended implementation did not make it easy to replace current frames with new updates. The

introduction a factory method was deemed necessary to reduce maintenance and encapsulate the frame instantiation.

2. The *Player* object contained within the client is the only instance that will ever be needed. Therefore there is no requirement for it to be freely accessible and modified. Enclosing it within a Singleton pattern was deemed best.
3. For the sake of simplicity, implementing the system on a distributed system was avoided. As a result, the Network Manager was not implemented and an instance of the Database manager was added to the client in its place. A discussion on how to best implement a distributed system is discussed in the Amazon Web Services (AWS) discussion section and the Network Communications (under Additional Information) section.
4. It was intended to implement a third party component *TeamSpeak* (in an effort to incorporate component based development) which would provide the functionality of voice communications over a network. Due to time constraints, this was dropped. In an actual implementation of a games match making framework, this would be an important feature to incorporate as it directly affects the user's interaction with his or her friends.
5. The Model View Controller was added so as to separate different layers of abstraction, i.e.: it was used to separate the elements that define our GUI and our 'world' objects (ref Fowler).

### Matchmaking Server Observations

The implementation of the Matchmaking server more or less corresponds to that specified in the class diagram, with the exception of a few functions for computations. The concurrency achieved allowed a process to handle many queues for many games, organising players according to their ELO. Exceptional effort was expended into adding 'computational value' to the skeleton code, i.e.: the server does a lot of work (evaluating players).

### Database Manager Observations

The implementation of the database had to be adjusted from the initial design as the practicality of passing certain operations particular arguments was impractical. While the interface to the database is almost trivial as it is, in effect, a series of accessor and mutator operations, this interface becomes exceptionally useful when implementing Java's Remote Method Invocation (RMI – see *Network Communication* under *Addition Information*).

## Implementation Criticisms

These are a series of criticisms within our current 'proof of concept' implementation.

### Client

In order to improve comprehension, we should have made use of enumerators in order to more clearly define what certain values meant. This is especially relevant when asking the factory for a particular frame.

### Matchmaking

1. Threads are not dynamically allocated a tier based off the proportion of players to tiers.
2. The number of threads is not dynamically allocated.
3. The matchmaker does not asynchronously update the client with queue information.
4. The matchmaker does not actually talk to a game server. Instead it creates a dummy result for a 'game' and relays this to the client.
5. Does not make use of the other processors (if any).

### Network Communication

1. There is no use made if the Java RMI API. This would have simplified the process by which operations on the database (which is presumably on another system over a network) could be invoked, and data retrieved.
2. There is no separation of network related operations from the client and server. Sockets are embedded (aggregated) within these.

# Additional Information

## Network Communication

### Overview

Designing a system that could communicate through a network was a central issue. The initial design, which incorporated a 'Network Manager', turned out to be infeasible. One generic method that accepted and/or returned an array of Objects introduced a lot of complexity and high maintenance.

The core issue was how to invoke methods on the server from the client's side. We knew we needed some sort of remote procedure call (RPC) but were unsure how to implement it. An asynchronous call to the server via sockets and threads, where the server accepted an integer and, using a switch case, called the appropriate method was the best solution devised. There was still the problem of accepting information and attempting to reduce maintenance costs.

Another solution was to mirror each method on the client/server side but this introduced vast amounts of code, inferring extreme maintenance, complexity and comprehension issues.

After researching better solutions, Java's Remote Method Invocation (RMI) service seemed the best route. Encouraging the programming to interfaces principle, RMI was considered the best solution. Unfortunately we could not implement RMI fully. By the time we figured out how to implement it, we had run too short on time. The coding samples are test fragments that make use of the database manager interface (represented by *I_RemoteDatabaseManager*).

A brief word on Java's RMI: Java RMI is a Java API that performs the object orientated equivalent of an RPC call. RMI can handle the transfer of serialised objects and allows for distributed garbage collection (ref Wikipedia).

The RMI design would combine the roles of the 'Network Manager' and the 'Database Manager' specified in the original class diagram.

Despite the usefulness of the RMI, there are still a number of issues. Handling varying levels of load per unit time, etc. introduces complexities beyond the scope of the RMI server. Elasticity, stress, high write throughput would all be factors in the implementation of the database management system. As such, Java RMI (the entirety of Java in fact) and MySQL are not really suitable for such an environment. Java's virtual machine introduces decreased performance and MySQL does not scale easily. The final solution is to integrate a third party service that allows the issues discussed to be dealt with.

# The Future - Serenity Gaming and Amazon Web Services

## Overview

Serenity gaming was conceptualised in order to aid gaming companies in the growth of their franchise and the expansion of their player base. As such we, the providers of this product have to both focus on our current client's needs but also prepare and build towards the future that we want and we must build expecting the best success we can achieve.

In light of that we should be expecting our software one day to be used by massive numbers of people. These figures are absolutely impossible to predict and even if they were, they have no guarantee of remaining static. Games are a fickle entity, one day a million players could be playing your game and the next merely hundreds. If that's hard to believe perhaps something more along the lines of one hour hundreds of thousands, the next hundreds, quite possible in certain peak states, for e.g. during the announcement of a special event within the game.

Likewise many games play out over iterations of patches and or expansions where towards the end of one patch or expansion numbers tend to dwindle off from the game as content becomes stale and increasing numbers of players have finished said content.

For these reasons something as simple as server purchases for our system becomes an incredibly hard task to complete. As we may buy massive bulks of server space that turn out perfectly useful at early game launch when interest is high but suddenly a few months later we are running that same server space for less than a quarter of the population it can handle.

Another issue that may occur or need to be brought into consideration for system design is location. Datacentres are not cheap to purchase and/or run and if we truly want to provide the best possible service to players to provide the best possible reputation for our client (the game company) we need to have location available to service players globally. It's no good having all the players in North America connecting through European servers and experiencing unwanted latency that their European companions have no issues with.

This is a quick and easy way to isolate entire continents of millions of potential customers from our client and would be massively detrimental to our company's vision of unifying players in a single highly enjoyable community experience.

There are even more heavy design decisions we must undertake, where should players be connected through, what gives them easiest access to the system, how will we handle security for our servers and databases, how will we handle failing servers or even entire failing datacentres in a region.

These are all time/resource consuming decisions to solve and even more consuming and costly to implement. All the time taking focus of our team away from our software development and more into architecture design, limiting resources available to producing the best possible program/application and spreading reources more into improving back ends and architecture.

Now it should be known this is not a cry for lack of investment in backend and system architecture, these are recognised as extremely vital portions of our system and rather than provide less attention to them, we believe a better approach is to take the smarter approach. That being said we have elected to use Amazon web services for the distribution of our system as it will massively help with all the topics discussed in this interview and much more. All of which we will cover in the next few sections in this topic of discussion, the future of Serenity Gaming.

## What is Amazon Web Services?

Amazon web services is a cloud computing platform made up of over thirty remote computing services also known as web services. These services are designed to work together to build sophisticated scalable applications. The services act as building blocks that can be built together and rebuilt to suit your current needs.

Amazon web services gives easy on demand access to highly durable storage e.g. Amazon Glacier, Amazon EBS and Amazon S3 which we use in our architecture concept for server implementation in our system shown later in this discussion, AWS also gives on demand access to low cost compute e.g. one of AWS best known web services Amazon EC2 (Elastic Compute Cloud) which is also used in our architecture concept.

High performance databases are also available from AWS examples such as Amazon Red Shift, Amazon ElastiCache, DynamoDB (NoSQL) and the database we will be using in the first draft of system architecture, Amazon RDS (MySQL).

The tools to manage all these resources are along with the resources are all available without upfront costs where you only pay for what you use. This is the true glory of amazon web services.

E.g. imagine you are developing a database driven application where high availability and low cost are important. You can use AWS to store your important documents and files with storage services designed for eleven lines of durability and power the app using your choice of relational or non-relational database.

You'll have reliable managed databases running in minutes across multiple geographically isolated datacentres for redundancy and availability. It is also easy to deploy applications across these databases easily through tools and languages of your choice as AWS provides integration support for tools such as Visual studio and eclipse, and also support for common languages such as Java, Python and PHP.

There is a large selection of computational resources to power any application with support for I/O, storage and CPU intensive workloads. Experimentation is easy and low risk with on demand access to a wide range for hardware configurations and flexibility to evaluate and run virtually any technology or tool

Your systems infrastructure can grow and shrink automatically with AWS as your needs change you can round out your application with load balancing (ELB discussed later), domain name services (Route 53 discussed later), global content delivery network(Cloud Front discussed later) and automate everything with a wide array of SDKs supplied through AWS.

Working with data through AWS is easy, with simple tools to integrate, import/export data, manage Hadoop clusters, spin up petabyte scale data warehousing, archiving frequently accessed information and build compliant secure environments which integrate into your existing infrastructure via private dedicated connectivity.

To summarize: AWS provides a complete cloud platform ready for use for virtually any work load e.g. Business Applications, building Games or Big Data. AWS gives access to resources you need to create sophisticated scalable applications of any size or shape.

## AWS Security

### Identity and Access Managers (IAM's)

Enables you to securely control access to AWS services and resources for your users. Using IAM, you can create and manage AWS users and groups and use permissions to allow and deny their access to AWS resources.

### Key Pairs

Amazon EC2 uses public–key cryptography to encrypt and decrypt login information. Public–key cryptography uses a public key to encrypt a piece of data, such as a password, then the recipient uses the private key to decrypt the data. The public and private keys are known as a *key pair*.

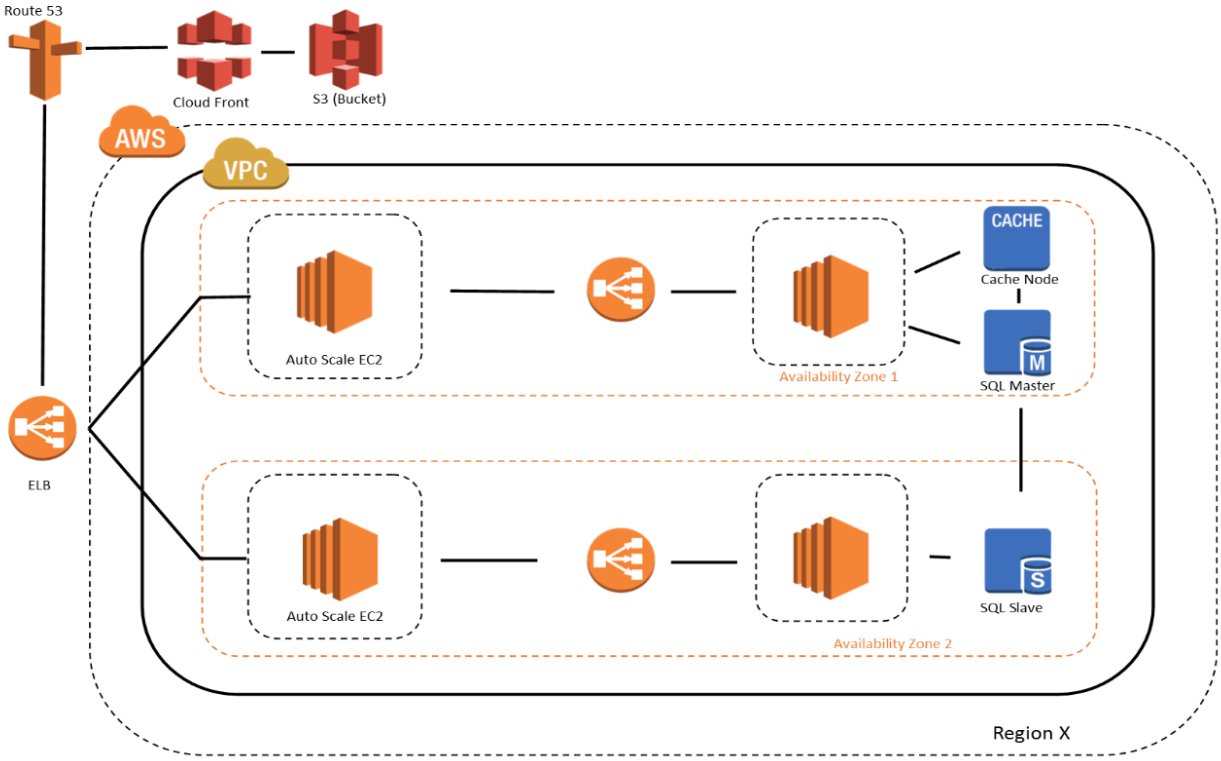
To log in to your instance, you must create a key pair, specify the name of the key pair when you launch the instance, and provide the private key when you connect to the instance. Linux instances have no password, and you use a key pair to log in using SSH.

### Security Groups

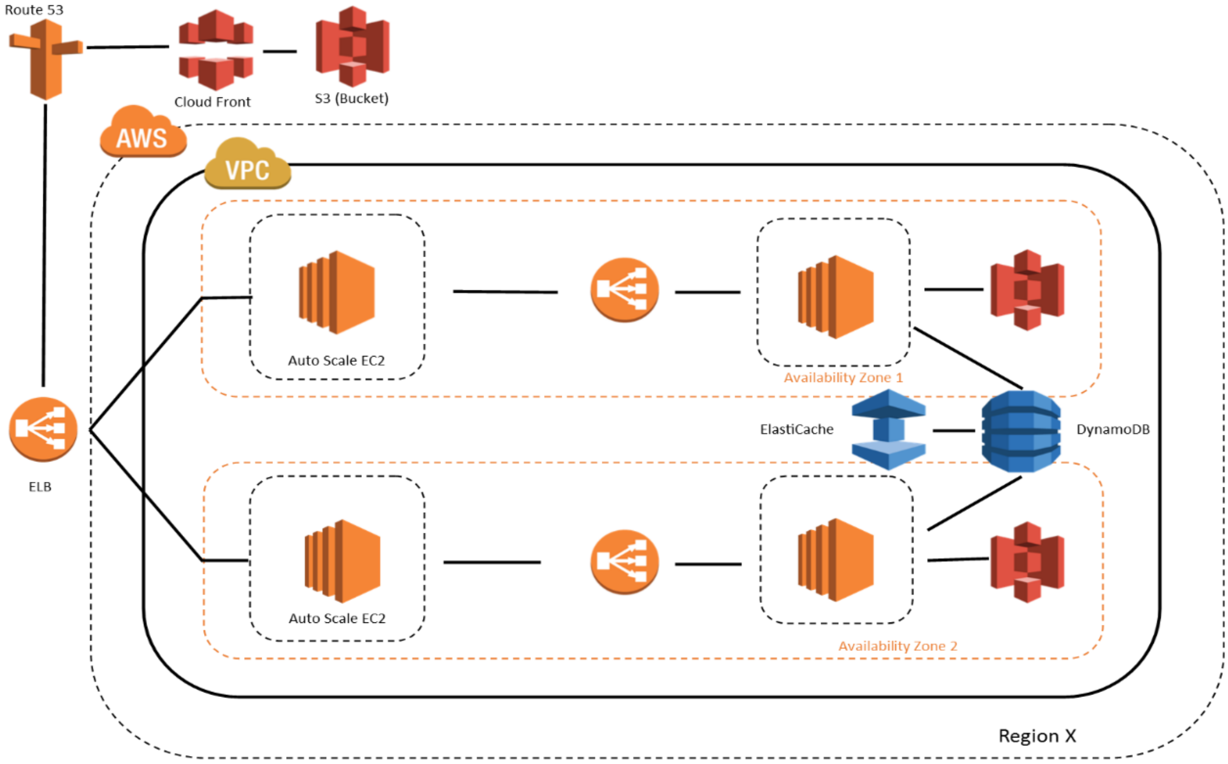
A *security group* acts as a virtual firewall that controls the traffic for one or more instances. When you launch an instance, you associate one or more security groups with the instance. You add rules to each security group that allow traffic to or from its associated instances. You can modify the rules for a security group at any time; the new rules are automatically applied to all instances that are associated with the security group. When we decide whether to allow traffic to reach an instance, we evaluate all the rules from all the security groups that are associated with the instance.

## Our Plan

### First Draft



First Iteration – Improved Scalability



The previous two diagrams show our plan of approach for handling information storage/retrieval and server implementation to account for a constantly resizing system. The alterations between first implementation and first iteration will be discussed further within the database sections below. We will also discuss all the other web services present in both diagrams, we will start with VPC.

Both the architectures account for dynamic changes to user flow and request for the system but also incorporate a static handling through cloud front and S3 bucket, to quickly and with low cost provide stored static content to the user.

## Virtual Private Cloud (VPC)

By Default when creating instances within AWS you are using the default networking. You are within the same “cloud” as everyone else in AWS, you are protected by your security groups (discussed later) but if you would like to run within your own private network VPC is the way to go.

VPC allows creation of VPNs (virtual private networks) effectively allowing complete cutting off of internet access to your “home” network where you can map to the instance connected to cloud but cloud cannot connect said instances back to you. It can be left completely up to you whether to open connections to the internet.

Effectively VPC is adding further security to an already secure system architecture. Amazon VPC enables you to use your own isolated resources within the AWS cloud, and then connect those resources directly to your own datacentre using industry-standard encrypted IPsec VPN connections.

Security is always a plus, but in particular for our project we use VPC to allow internal Elastic load balancing to improve IP security within system (explained further in ELB section).

## Route 53

Amazon Route 53 is a highly available and scalable cloud Domain Name System (DNS) web service. Route 53 provides a lot of benefits for routing traffic and is also very cost effective. It works on latency based routing so it analyses incoming traffic in real time and determines fastest edge location to send users to rather than shortest distance. It effectively connects users to infrastructure running in AWS, in our case to cloud front or to the ELB, for static client data/server requests respectively.

You can use Amazon Route 53 to configure DNS health checks to route traffic to healthy endpoints or to independently monitor the health of your application and its endpoints. Amazon Route 53 makes it possible for you to manage traffic globally through a variety of routing types, including Latency Based Routing, Geo DNS, and Weighted Round Robin—all of which can be combined with DNS Failover in order to enable a variety of low-latency, fault-tolerant architectures.

## Elastic Load Balancer (ELB)

As shown in our plan diagrams a single region within AWS can contain multiple availability zones, the idea behind this being to provide ability to keep traffic in a single zone stable. Elastic Load balancer is what assists this functionality in AWS.

It is able to detect unhealthy zones in a region and reroutes traffic to the remaining healthy ones in the region. If all instances in a single zone are unhealthy, ELB reroutes to an instance in another healthy availability zone in our region.

ELB is as its names suggests elastic so it can scale its request handling to meet varying levels of traffic to our system without the need for us to intervene manually.

A great example of the use of ELB to sort issues above was in 2012 during the Mars Rover landing, NASA was going to stream the event to the world through a website. This was one of the largest events in history and they had no possible way of calculating the numbers that would hit their site during the event and also could not use their current infrastructure as it was not designed for the numbers that they could guess at.

So they implemented a scalable solution through AWS incorporation the ELB which allowed a successful streaming event even at peak of 120k or more viewers requesting video stream data without causing server response issues. Full details and graphs available in reference here.

Finally ELB can even help improve the security of our system. By running internal ELBs within an availability zone, this only works within VPC. We can implement a multi-tiered architecture using internal and internet-facing load balancers to route traffic between application tiers. With this multi-tier architecture, our application infrastructure can use private IP addresses and security groups, allowing us to expose only the internet-facing tier with public IP addresses.

## Cloud Front

Cloud Front is a global content delivery web service. We use it within our system with an S3 bucket as server to provide users with the static content of our system such as sound files, images, web pages pulled into client etc.

## Simple Storage Service (S3)

Amazon S3 is used in both system architecture diagrams as a server to our cloud front web service to implement a static interface provider to the user, where S3 stores web pages, sound files, videos, pictures etc.

S3 is secure, durable and highly scalable. Also we only pay for the storage we use, there is no minimum fee so it is cost effective.

## Elastic Compute Cloud (EC2)

EC2 is one of AWS's main services. EC2 provides resizable compute capacity in the cloud. EC2 allows us to spin up instances windows or Linux in a matter of minutes that are reliable, secure and inexpensive.

We have designed our architecture to have multiple availability zones of auto scaling EC2 instances internally routed through ELBs to internal EC2 instances (as explained in ELB section) for both increased security but also reliability and scalability. Auto scale groups do just what they say and allow creation of new EC2 instances within a region when required – I.e. all current instances in availability zone unhealthy from traffic overload.

For this reason we will need to take into account creating our application to be able to work within an auto scaling environment, but the cost of resources to implement this feature is worth the payoff for the ability to have such a flexible system incorporated. It should be noted auto scale groups allow the number of instances to shrink as well as grow.

## Databases – Why the iteration change

When devising the first draft for this system architecture MySQL database was chosen out of comfort of working with MySQL all the time in the past.

Amazon RDS was the web service chosen. It handles both backups and caching automatically for us. RDS is automatically placed in separate security group to webserver so adds added protection on our database.

Finally in this implementation we required the DB to be highly available especially at times of high traffic. For this reason we set up a system of direct caching and polling of slaves to master database in other availability zones in a region. This allowed two things first faster response of frequent queries from cache and second slave databases with replicated data from master available to zones.

So why the change. Well even with the approaches to improve functionality to our database in first draft, MySQL is just not effective when it comes to scaling. This is where DynamoDB comes in. DynamoDB is a NoSQL database so avoids the faults MySQL suffers at high user numbers, and has the extra bonus of providing predictable performance with seamless scalability. We use a cache again in this case to improve functionality as in a game hosting service the will definitely be a lot of common database read requests (view leaderboards e.g.) Caches help a lot with this.

In the case of the first iteration we use elastiCache, which automatically detects and replaces failed nodes, reducing the overhead associated with self-managed infrastructures and provides a resilient system that mitigates the risk of overloaded databases, which slow website and application load times.

## Conclusion

We have created a highly scalable, cost effective, secure, reliable and efficient architecture through AWS with very little time input required to design proposed system. All services are available directly through AWS console and are run through very intuitive interfaces.

We also have the opportunity of testing AWS through free tier option which provides high quality service access at fair monthly quotas.

Even on full tier access, when we run our system using AWS design we only pay for what we use, and what we use is scaled to exactly what we need. There is no excess and we can rely on the service availability.

# References

**Cite it Right, A guide to the Harvard referencing system:** http://www3.ul.ie/referencing/miniSite/citeItRight.htm

## Website:

- Statista (publication date unknown) Market volume of online gaming worldwide [online], available: http://www.statista.com/statistics/270728/market-volume-of-online-gaming-worldwide/ [November 2nd 2014]
- Wikimedia Foundation Inc. (2007) V-Model (software development) [online], available: http://en.wikipedia.org/wiki/V-Model_(software_development) [October 30th 2014]
- Wikimedia Foundation Inc. (2009) Waterfall model [online], available: http://en.wikipedia.org/wiki/Waterfall_model [October 30th 2014]
- oodesign.com (unknown), Proxy Pattern[online], http://www.oodesign.com/proxy-pattern.html [19 Oct 2014]
- stackexchange.com (01 Sept 2011), Why Should I Use a MVC Pattern? [online], http://programmers.stackexchange.com/questions/105352/why-should-i-use-an-mvc-pattern [20 Sept 2014]
- oracle.com (2014), java.rmi.Remote Interface Remote [online], https://docs.oracle.com/javase/7/docs/api/java/rmi/Remote.html [16 Nov 2014]
- oracle.com (2014), java.rmi.Registry Interface Registry [online], https://docs.oracle.com/javase/7/docs/api/java/rmi/registry/Registry.html [16 Nov 2014]
- msdn.microsoft.com (2013), UML Sequence Diagrams [online], http://msdn.microsoft.com/en-us/library/dd409389.aspx [23 Sept 2014]
- ibm.com (2004), UML Basics [online], http://www.ibm.com/developerworks/rational/library/content/RationalEdge/sep04/bell/ [29 Sept 2014]
- playdota.com (2013), How Match Making Systems Work [online], http://www.playdota.com/forums/blog.php?b=148286 [29 Sept 2014]
- http://aws.amazon.com/ (2014) Amazon Web Services [online], available: http://aws.amazon.com/ [November 2nd 2014]
- youtube.com (2012) AWS re: Invent CPN 205: Zero to Millions of Requests [online], available: https://www.youtube.com/watch?v=xKF-Aawz9oc [October 25th 2014]
- youtube.com (2014) What is Amazon Web Services [online], available: https://www.youtube.com/watch?v=mZ5H8sn_2ZI [October 25th 2014]

## Pictures

- Hostventura (publication date unknown) Development methods we follow [online], available: http://www.hostventura.com/en/menu/development/methodology#agile [November 2nd 2014]