AWESOME GAMING

CS4125: System Analysis and Design

Steve Harte 09014605
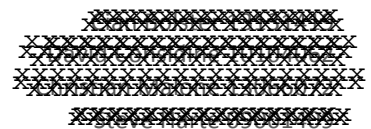
CS4125: Systems Analysis and Design. Semester I, 2015-2016
Guidance on the MARKING SCHEME for Team-Based Project:
Version 1 (29th September 2015 - Week 4)

Name: xxxxxxxx	ID: xxxxxxxx
Name: xxxxxxxxxxxxx	ID: xxxxxxxx
Name: xxxxxxxxxxxxx	ID: xxxxxxxx
Name: xxxxxxxxxxxxx	ID: xxxxxxxx

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

Contents

Narrative Description

Overview

With advances in technologies in the last several years, the videogames industry has seen an ever increasing shift towards online play. This has presented itself as a huge increase in video game releases from genres such as MMORPG (Massively Multiplayer Online Role-Playing Game) and competitive online multiplayer. The majority of video games released today include online functionality in some form (e.g. competitive multiplayer, cooperative play, spectating and online leader boards). With this has come a need to support a social aspect to the gaming experience. With most online games involving players who interact with other players, there is an expectation that services to enhance the social aspect (voice chat, messaging, friend finding, matchmaking, forming parties, etc.) come packaged with the game. Sadly this is not always guaranteed or they are not of a very high standard.

Our aim is to provide these social services to our users at a very high standard of quality. For every independent game company or publisher that avails of our service, their players will have a dedicated independent platform where they can chat with one another through voice chat and messaging, organise friends and communities, and make parties with other players to play with before starting up their game of choice.

What this means for game developers is they no longer have a need to try and develop these features for their games. Instead, they can focus on developing their game with the peace of mind that they can let our services handle the social aspects for them.

What this means for the players is that they will have a single uniform platform that provides them with all the social features they would expect included with their game of choice. This eliminates the cumbersome task of having to learn a different set of rules to operate similar features that might have been implemented very differently from game to game.

What is "Awesome Gaming"?

Awesome Gaming is a game independent client program that allows players to easily communicate with friends, communities and other players with similar interests. It acts as the social network to a company's game allowing players to effortlessly interact with one another regardless of what game they are playing.

Software Life Cycle

After analysis of the software requirements and various approaches to software life cycle management it was decided the Agile Development model would best suit our system and requirements. We discussed the pros and cons of other approaches, such as the Waterfall model, and the V- Model.

Waterfall:

Pro:	Con:
Easy to understand and implement.	Doesn't reflect iterative nature of exploratory development.
"Define-before- design, design-before-code."	Difficult to integrate risk management.
Easily Identifiable Mile-Stones	Inflexible.

While we approached this project in the beginning unintentionally following a very Waterfall like methodology, as the group met and refined our design process and supporting documentation our approach and methodology changed. We found the Waterfall approach to be too inflexible for our needs.

Figure: W.M., adapted from: (RainsHorde 2015)

The classic waterfall development model
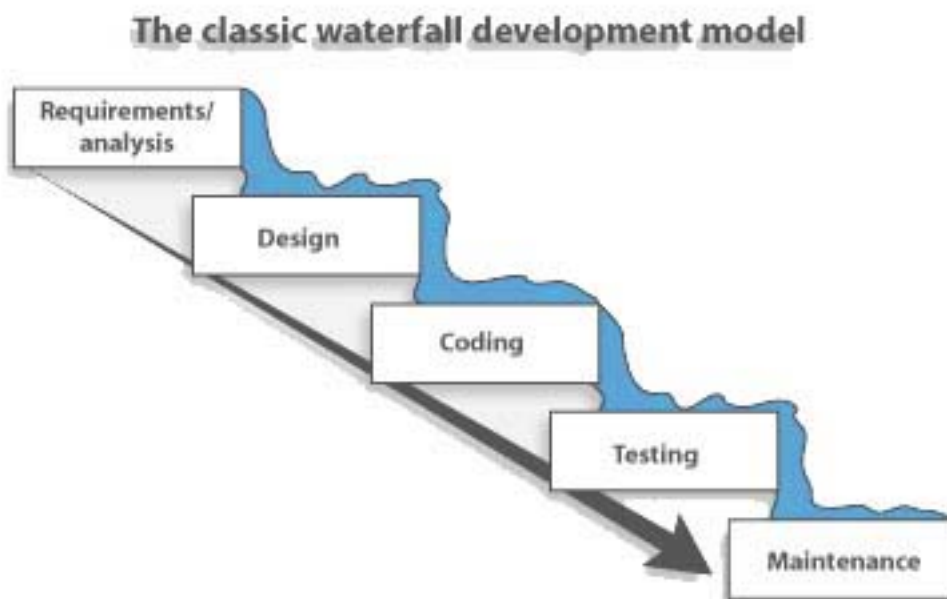
V-Model:

Pro:	Con:
Simple and easy to use.	Very rigid like the waterfall model.
Each phase has specific deliverables.	Adjusting scope is difficult.
Works well for small projects.	No Early Prototypes.

Due to the inflexibility and lack of early prototyping this approach was deemed unsuitable for what we required. As our design was constantly being reassessed this approach was deemed too inefficient.

Figure: V.M., adapted from: (Business Analyst Mentor)
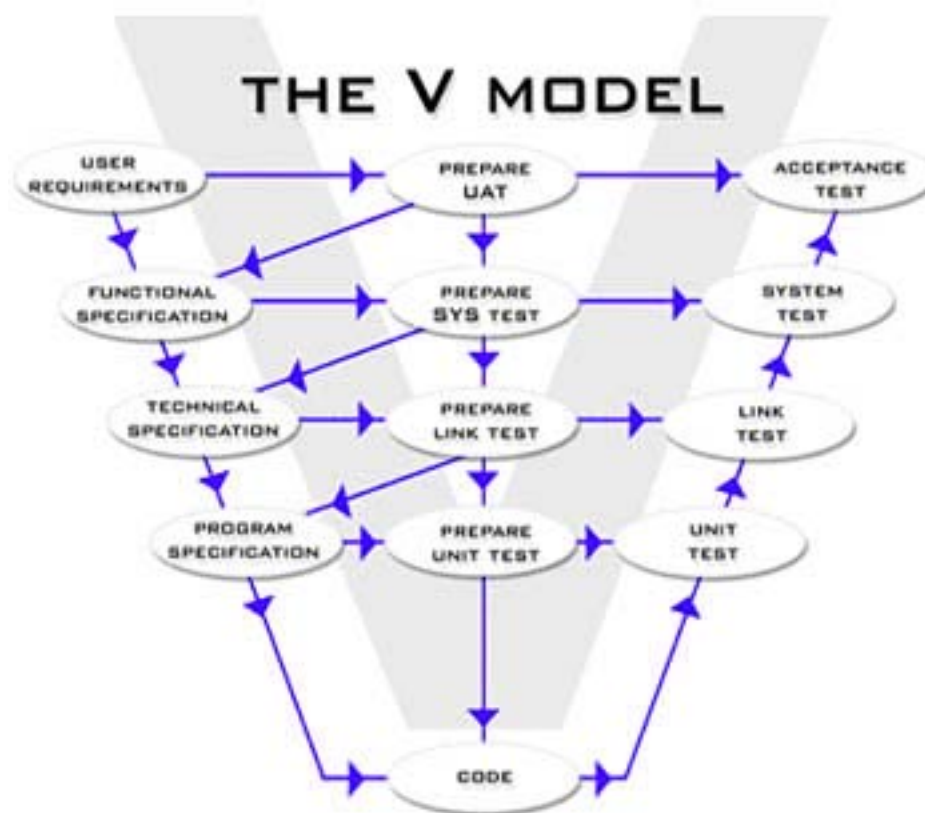
Agile:

Pro:	Cons:
Lightweight methods.	Difficult to scale up to large projects.
Iterative.	Programming pairs is costly.
Produces good team cohesion.	Needs experience and skill if not to degenerate into code-and-fix.

Due to the flexibility offered by the agile approach it was decided this was methodology that matched our needs the most. This lent to an iterative approach to our system, enabling us to refine our methods and processes as development progressed. Our first iteration of the system can be seen in Appendix A. As can be seen in a quick comparison of this and the final code of our system, the refinement of our processes can be seen. Each iteration of the system was a further improvement of our implementation of observer and controller functions and development processes.

Figure: AGM, adapted from: (AplicaTech 2015)
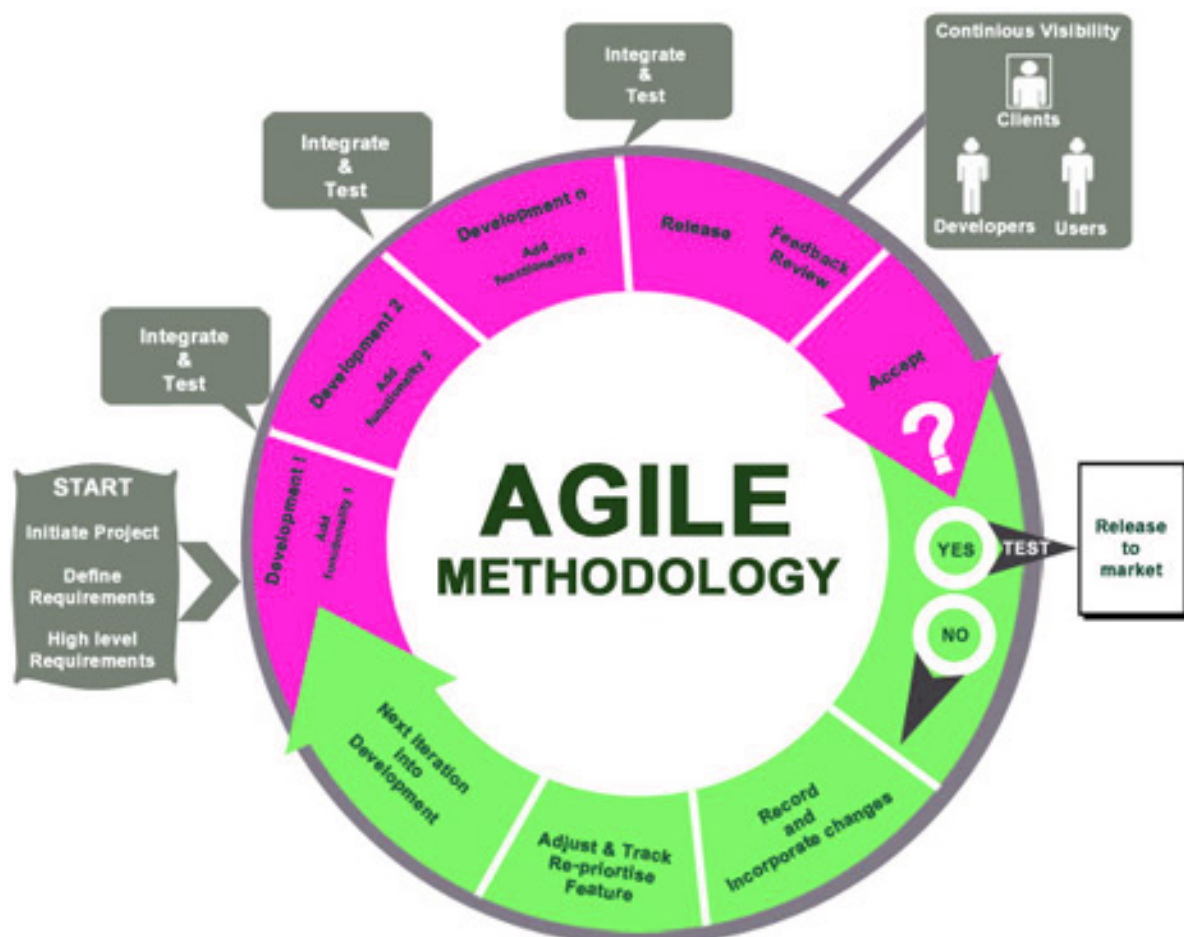
Project Plan

Job Name	Job Description	Allocated to	Week
Narrative	Narrative description of business scenarios	XXXXXX	5
Presentation	Company Logo/Design Cover Page	XXXXXXX	5
Software Life Cycle	Discussion of the Software Model Used	XXXXX	5
Project Plan	Specifying Jobs and Roles	XXXXXX	4
Requirements	Use Case Diagrams	XXXXX	6
	Use Case Descriptions	Group	6
	Structured Use Case Description	XXXXX	6
	Non-Functional Requirements	Group	6
	Tactics for handling Quality Attributes	XXXXXXXXX	6
	Screenshots of GUI	XXXXXXXXX	12
System Architecture	Architecture Diagram with Interfaces	Group	8
Analysis Sketches	Identify Candidate Classes	Group	6
	Class Diagrams	Group	7
	Communication Diagrams	Group	7
	Entity Relationship Diagrams	Group	7
Code	Code Implementation	Group	8-12
Design	Architectural Diagrams	XXXXXX	12
	Class Diagrams	XXXXXXXX	12
	Interaction Diagram	XXXXXXXX	12
	State Chart	XXXXXXXX	12
	Description of Patterns	XXXXXXX	12
	Approach to Concurrency Support	XXXXXX	12
Critique	Critique on quality of Analysis and Design	Group	12
References	Sources used for learning/information	Group	4-12

Requirements

Functional Requirements

Use Case Diagrams:
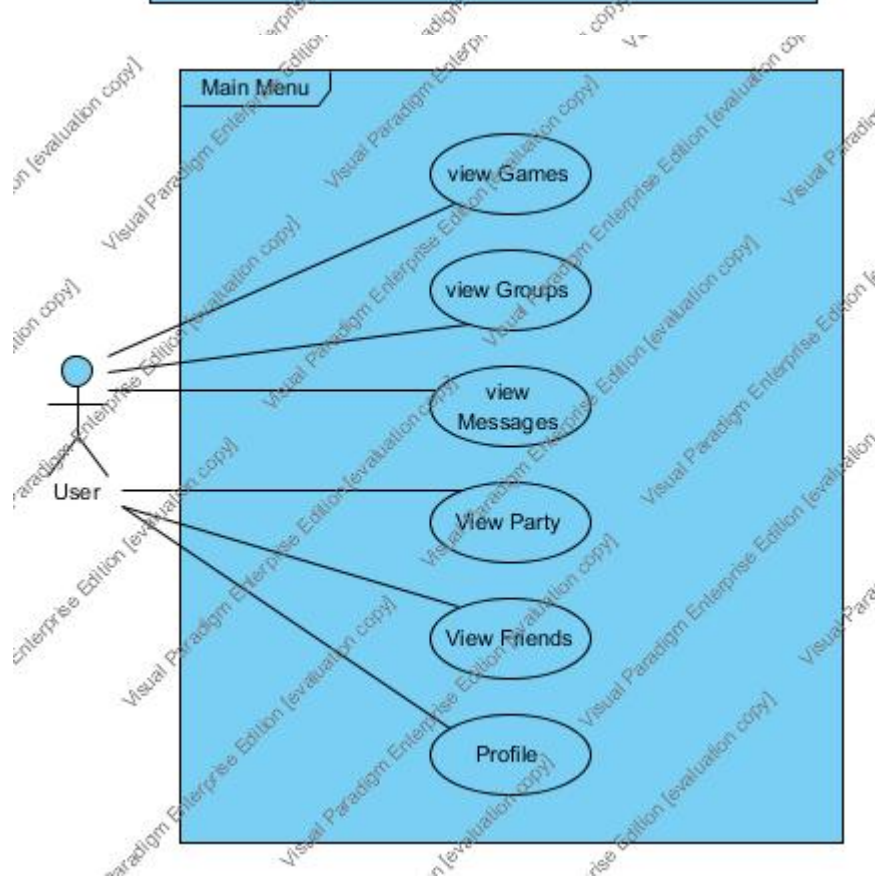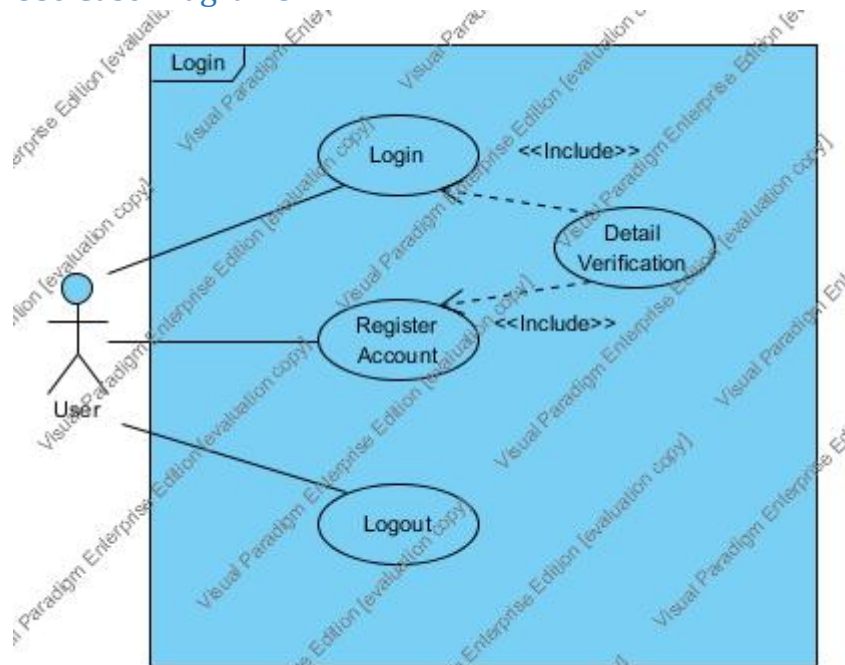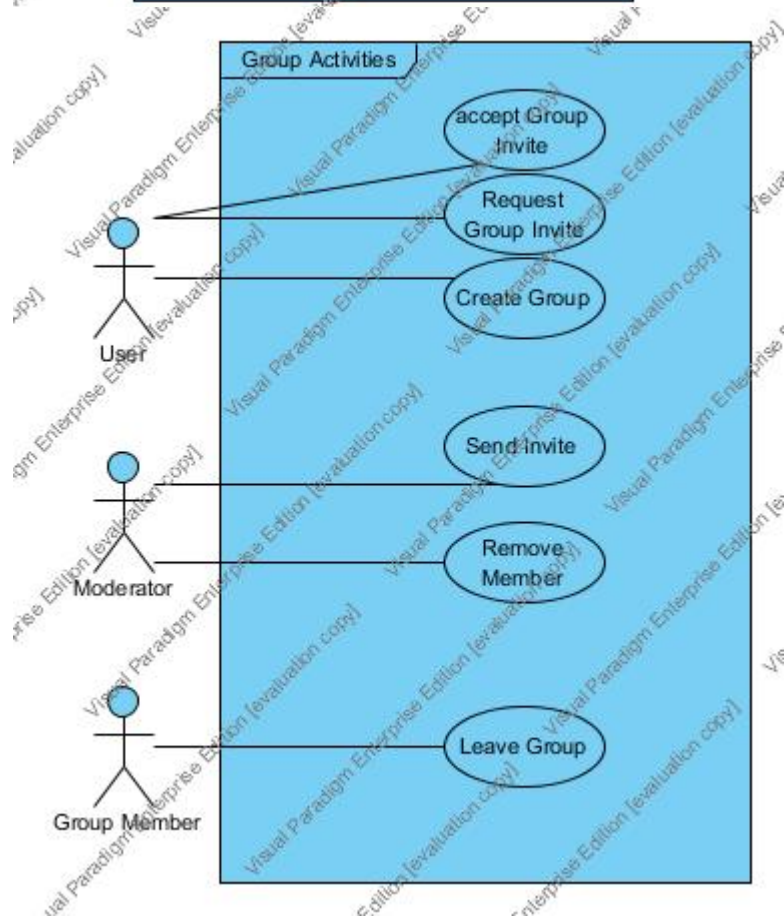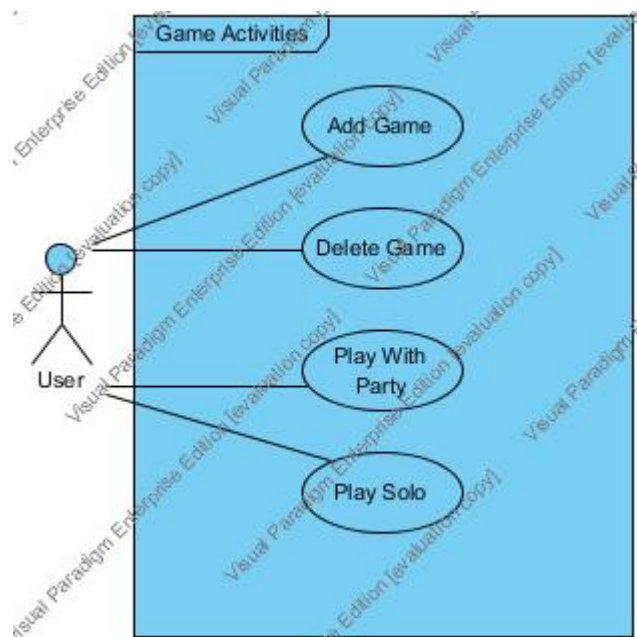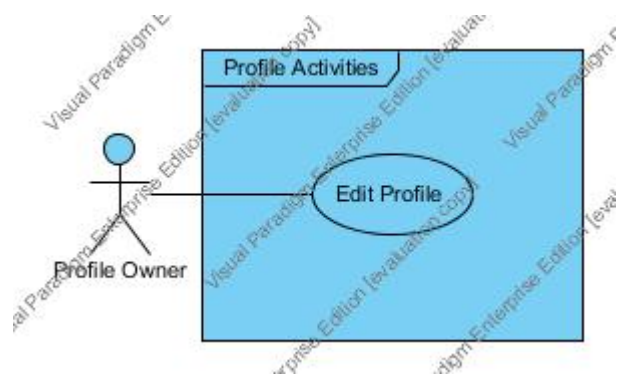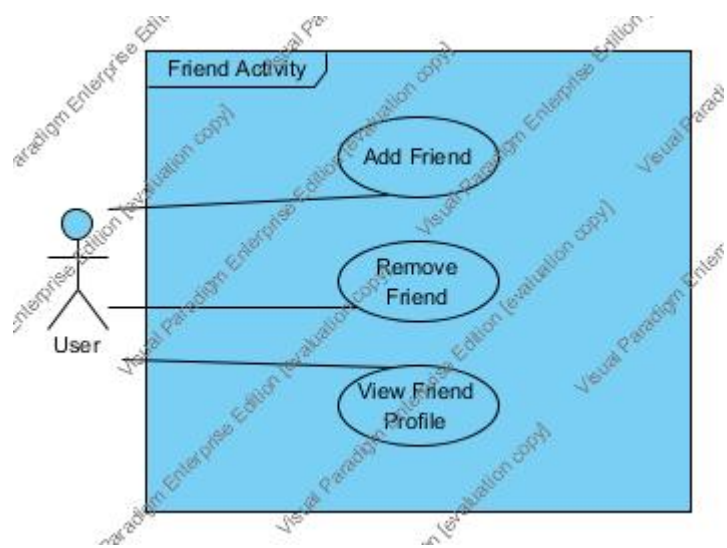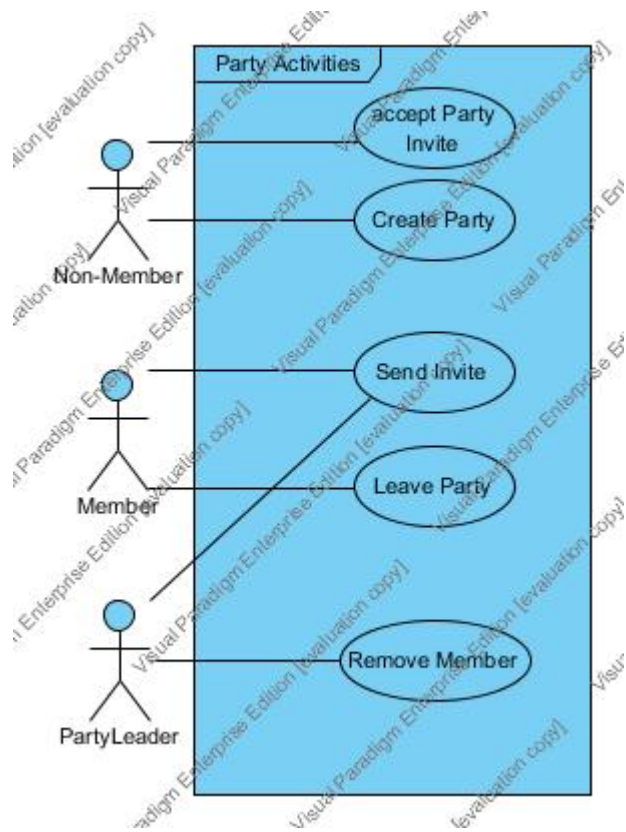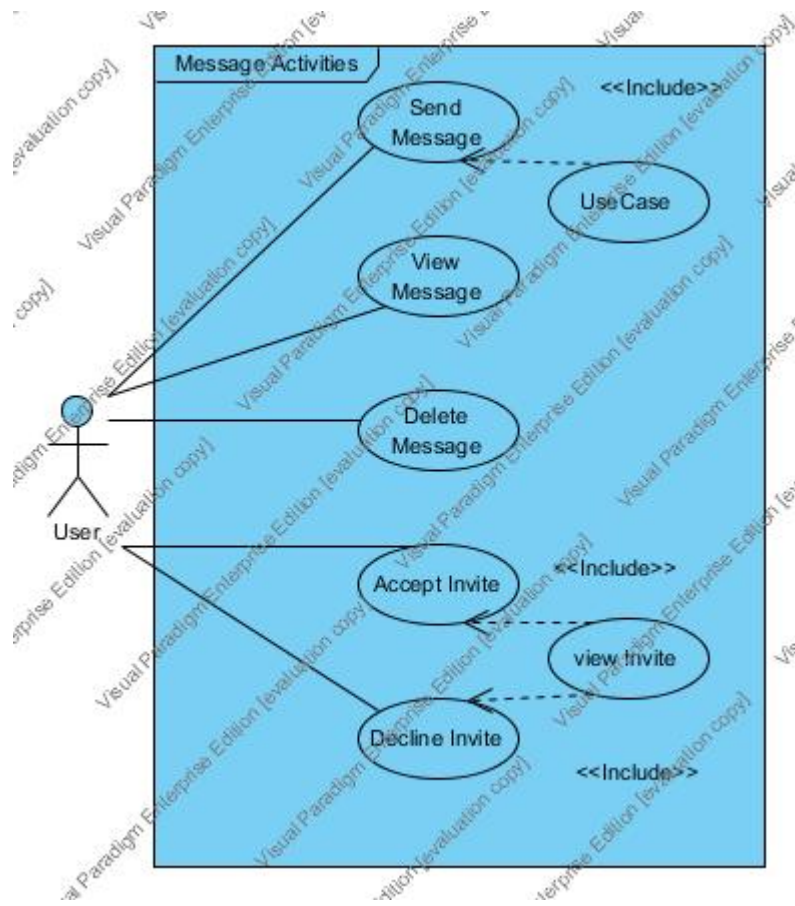
Structured Use Case Descriptions:

Use case 01: Login

Actor Action	System Response
1. Enters Username and Password into designated fields and selects "Login"	2. Authenticates Username and Password. Displays home screen
Alternative course: 4a. Authentication fails. System displays error message.	

Use case 02: Register New Account

Actor Action	System Response
1. Selects "Register New Account"	2. Displays registration form on screen
3. Fills in details and selects "Enter"	4. Validates user details. Sends verification email to users email address.
5. Selects acknowledgement link in confirmation email.	6. Stores user data, and registers user. Launch Home screen for first time user

Alternative course:
    4a. Validation fails. System displays error message.
    4b. System sends verification message via sms message.	

Use Case 03: Logout

Actor Action	System Response
1. Selects "Logout"	2. System updates and saves user's session. User is set to offline

Use case 04: Add New Game

Actor Action	System Response
1. Selects "Add Game".	2. Displays search window
3. Selects game's executable file	4. Adds selected game to library of accessible games.

Alternative course:
    3a. User does not add game. Selects "Cancel"
    4a. Game is not compatible. System displays error message	

Use Case 05: Delete Game

Actor Action	System Response
1. The actor highlights game and selects "Delete Game"	2. Displays message "Are you sure?"
3. Selects "Yes"	4. Removes game from list of games
Alternative course: 3a. Actor selects "No" 3a.1. Game is not removed from list of games	

Use case 06: Play with Party

Actor Action	System Response
1. Select "Play with party"	2. System launches game.

Use Case 07: Play Solo

Actor Action	System Response
1. Actor selects "Play Solo"	2. System launches the game.

Use Case 08: Accept Invite to join Community Group

Actor Action	System Response
1. Select "Accept invite"	2. Displays message asking for confirmation of decision
3. Selects "Accept"	4. Community group is updated with user added, users profile is updated to reflect they have joined the group

Use Case 09: Request Invite to join Community Group

Actor Action	System Response
1. Select "Search"	2. Displays search window
3. Enters Group name or "associated tags". Selects "Search"	4. Searches Database for groups associated with text data entered by Actor. Displays list of all Groups that meet search conditions.
5. Highlights Group and selects "Join Game"	6. Displays confirmation message. Sends request notification to Group Moderator

Use Case 10: Leave Community Group

Actor Action	System Response
1. Select "Leave Group"	2. Displays message to user to confirm action.
3. Selects "Confirm"	4. Displays confirmation message to user. Updates Community's member list.
Alternative course: 3a. User selects "Cancel" • User remains a member of the Community Group	

Use Case 11: Create Community Group

Actor Action	System Response
1. Select "Create new Group"	2. Display community group creation form
3. Enters community group details and submits	4. New group entry added to server database and confirmation email sent to user. User assigned as moderator
Extensions: After step 4, the user invites friends from friend list to join created community group. Alternative course: 3a. Group name entered already belongs to pre-existing group. • System displays error regarding non-unique information entered	

Non Functional Requirement: Security

● User's information should be encrypted and stored in database

Use Case 12: Send Community Group Invite

Actor Action	System Response
1. Selects "Send invites"	2. Displays list of user's friends
3. Select friend(s) and select "Confirm"	4. Sends Group Invite(s) to selected user(s)
Alternative course: 2a. User currently has no friends. • System displays "No friends found" message. 3a. User selects "Cancel"	

Non Functional Requirement: Performance

● System should complete sending requests in under 5 seconds

Use Case 13: Remove Member from Community Group

Actor Action	
1. Double clicks Group name from list of Groups	2. Displays list of group members
3. Highlights member name and selects "Remove Member"	4. Displays message to user to confirm action.
5. Selects "Confirm"	6. Displays confirmation message to user. Removes selected user from Community's member list.
Alternative course: 1a. User highlights multiple members and selects "Remove from Group" 3a. User selects "Cancel" • User can select different member	

Use case 14: Send Message

Actor Action	System Response
1. Select "Create Message"	2. Displays list of user's friends.
3. Highlights friend username and selects "Confirm"	4. Displays input dialog for message.
5. User inputs message and selects "Send"	6. System sends message to the user that was selected from friend list.
Alternative course: 2a. User currently has no friends. • System displays "No friends found" message. 3a. User selects "Cancel" 5a. User selects "Cancel"	

Use case 15: View Message

Actor Action	System Response
1. User double clicks message from message list	2. Displays message
Alternative course: 1a. User currently has no messages. • System displays "No Messages found" message	

Use case 16: Delete Message

Actor Action	System Response
1. User highlight message and selects "Delete"	2. Displays message to user to confirm action.
3. User selects "Confirm"	4. Deletes message from database. Updates user interface
Alternative course: 1a. User currently has no messages. • System displays "No Messages found" message 5a. User selects "Cancel" • System closes dialog and returns to View Message	

Use case 17: Accept Friend Request

Actor Action	System Response
1. User selects a message	2. Displays invite
3. Selects "Accept"	4. System displays confirmation message. System added friend to friends list. System sends confirmation message to friend

Non Functional Requirement: Performance

- System should update the user's friend list in under 5 seconds

- Friend's system should receive confirmation message and update friend's friend list in a similar timeframe

Use case 18: Decline Friend Request

Actor Action	System Response
1. User selects a message	2. Displays message
3. Selects "Decline"	4. Displays message to user to confirm action.

Use Case 19: Create Party.

Actor Action	System Response
1. Actor selects "Create Party"	2. Displays available friends to invite to party.
3. Actor selects friends he/she wishes to invite	4. Creates party and sends invites to selected friends.
Extension: After step 4 System alerts the Actor when a friend has joined party or if they have declined the invite.	

Use case 21: Leave Party

Actor Action	System Response
1. Select "Leave Party"	2. Displays message to user to confirm action.
3. Selects "Confirm"	4. Displays confirmation message to user. Removes user from party member list.
Alternative course: 3a. User selects "Cancel" • User remains a Party member	

Use case 22: Send Invite to Party

Actor Action	System Response
1. Select "Invite to party"	2. Displays list of user's friends.
3. Select friend(s) and select "Confirm"	4. Displays confirmation message to user. Sends invite to friend
Alternative course: 3a. User selects "Cancel"	

Use case 23: Remove Party member

Actor Action	System Response
1. Highlight user on party roster and select "Remove from Party"	2. Displays confirmation dialog to user
3. Select "Confirm"	4. Removes user from party member list.
Alternative course: 2a. No members in party • System displays "No members in Party" 3a. User selects "Cancel"	

Use Case 24: Add Friend

Actor Action	System Response
1. Actor selects "Add Friend"	2. Asks the user for the username or email address of the user they want to add.
3. Actor inputs either the username or email address.	4. Sends an invite to the user with that username or email address
Extensions After step 6 the user could receive a notification if the invite was accepted or declined. Alternative course: 4a. Username or email address not found. • Displays message "User does not exist" and goes back to step 2.	

Use Case 25: Remove Friend

Actor Action	System Response
1. Actor selects "Remove a friend"	2. System lists out friends
3. The actor selects the friend he/she wishes to remove. Presses "Remove from Friends"	4. Displays message "Are you sure?"
5. Actor Selects "Yes"	6. Friend is removed from Friends list
Alternative course: 5a. Actor selects "No". 6a. Friend is not removed from Friends List and system goes back to step 2.	

Use case 26: View friend's Profile

Actor Action	System Response
1. Select friend from friends list	2. Displays friend's username
3. Select "View profile"	4. System displays friend's profile
Alternative course 1a. User has no friends 3a. User selects different friend 3b. User exits window	

Use case 27: Edit Profile

Actor Action	System Response
1. Select "Profile"	2. Displays user's profile fields as editable text boxes containing current details
3. User makes edits to information and Selects "Confirm"	4. Displays confirmation message to user."Profile updated"
Alternative course: 3a User selects "Cancel" • System displays confirm dialog • user selects "Confirm" • System exits Edit Profile 3b User selects "Cancel" • System displays confirm dialog • user selects "Cancel" • System returns to the editable profile	

Non Functional Requirement: Performance

- System should update the user's profile information in under 5 seconds
- User's information should be encrypted and updated in database

Non Functional Requirement: Security

- User's password information should be encrypted and updated in database

Use Case 20: Accept Invite to join Party

Use Case 20	Accept Party Invite	
Goal in Context	Service user accepts invitation to join a party	
Scope & level	Service server, Primary Task	
Preconditions	User has a registered account with the service. User is logged into their account. User has at least one outstanding Party Invite	
Success End Conditions	User joins Party	
Failed End Conditions	User is not a member of the Party	
Primary, Secondary Actors	Service user, service user who sent invite	
Trigger	User views Party invites	
Description	**Step**	**Action**
	1	User selects "Accept"
	2	System adds user to database storing the specific party's data
	3	System notifies Party creator
Extensions	**Step**	**Branching Action**
	1a	User declines invitation
	4a	User enters game with fellow party members
Variations		**Branching Action**
Related Information	20. Accept Party Invite	
Priority:	Top	
Performance	< 5 second for user to be added to Party	
Frequency	Over 1000/day	
Channel to actors	Interactive, Database	
OPEN ISSUES	What if party is in game already? What if invite is rescinded?	
Due Date	Release 1.0	
…any other management information…		
Superordinates	Send Party Invite (use case 22)	
Subordinates	Play Game with party (use case 06)	

Discussion on Tactics to support Quality Attributes:

We decided our service would be written using the Object Oriented Language Java. We decided on this due to your familiarity with the language, and the ease of portability across different platforms. As Java is one of the most widely used languages today, it makes increasing our team size easier at a later date, as there are more qualified programmers available.

As we have no real graphical requirements, apart from making a visually pleasing, and easy to use application, we decided there was no need for the extra graphical power offered by other languages like C++. While C++ might be preferable if we were doing more graphical rendering, due to the speed offered, our application will not require this.

We want to make our software user friendly and highly portable. With the availability of a java virtual machine for all systems means this language was the clear winner, as we do not need to compile it for each different individual operating system and different architectures.

Our user interface should be easy to use for those unfamiliar to our application so as not to scare new users away by unfamiliar or hard to follow UI. We want to make our application accessible to any user who so wishes to use it. As such, the low system requirements to run a Java application gives Java a further edge.
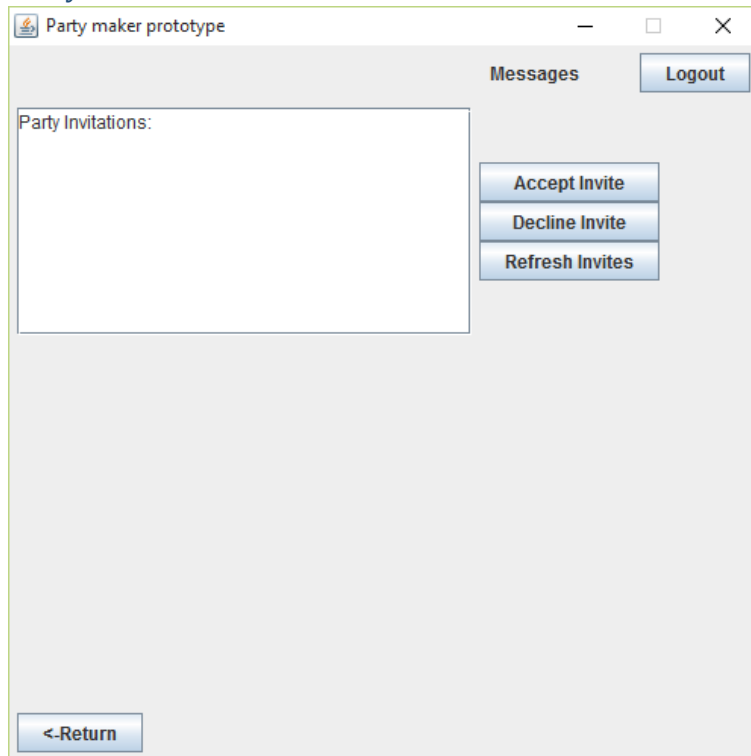
Our user interface shall be clear, and straightforward to follow, have a visually pleasing layout, and be simplistic in so far as it shall be easy to understand. Not simplistic in functionality.

Our server shall be monitored for traffic using a cloud based service like Monitis, we had looked at using a software based solutions but found the cost, and time required to upskill in understanding the use of the software too extensive. We looked at using Hyperic HQ, but found it falls short in some areas like automatic corrective actions, and the scale of manual effort required to run this remediation feature. During this phase we came across the white paper : Liberating Time :How Cloud based monitoring is transforming IT manager productivity. This paper clearly shows some of the benefits of using a software like Monitis, and the pitfalls that can happen, like reliant on network connectivity.

We shall also have a number of back up Databases which shall be kept separate from the base server so we can easily hotfix data issues as they arise.
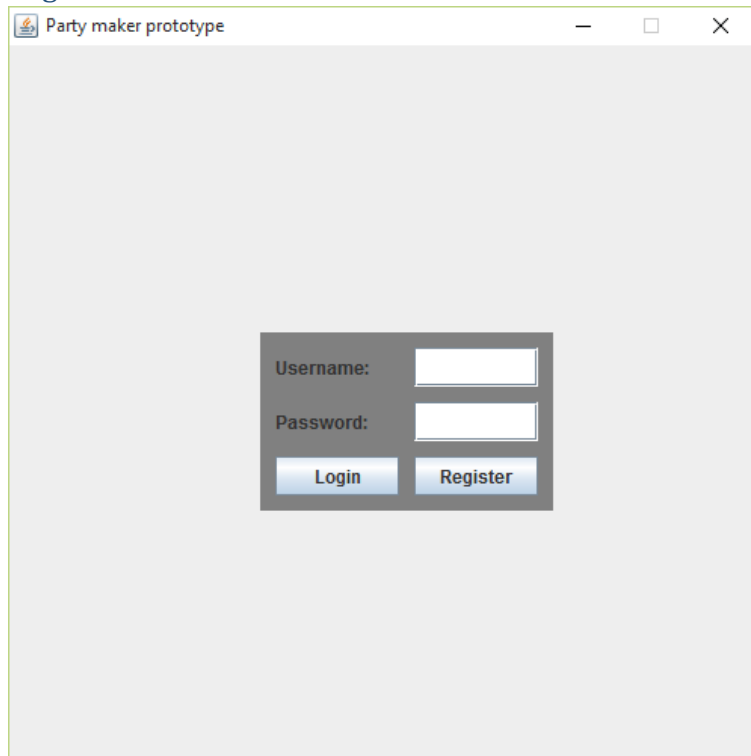
Screen Shots:
Party Invites Window

Login Window:

Main Menu:

Party Window:

Register Window:

Remove Player from Party:

System Architecture

Discussion

Architectural Decisions Taken:

We split the system into three sub-systems for our implementation: database, user interface and, session. Each sub-system was designed in a way to allow for the maximum amount of maintainability and portability for the whole system. The server-database would run on a separate machine that would allow the client system's session to interact with it.

The client system's session handled all the business logic and interactions with the database. This allowed the business logic to be separate from our user interface. This made for more maintainable software, which was our original aim.

The factory method can be seen in both the user interface system and the session system. Another pattern that we choose to utilise was the observer, which handled updating our changes to the information being displayed.

UML Workbench:

There was an initial large amount of research put into which UML workbench would be most appropriate workbench to be used. We found several including Visual Paradigm, Glify, Smart Draw and the UMLet plugin for eclipse. We found that interchanging between different workbenches for their strengths and weaknesses.

Visual Paradigm was the primary workbench that we used due to the fact that it was the most comprehensive and produces clearer and better quality diagrams. We originally used the trial of the enterprise edition as the format in which it was downloaded was easier for one of our members. Once the trial was over we had to switch to the community edition for the last of our diagrams. We had originally planned to use Glify for our ER diagram however we found it lacking and found a better tool in Smart Draw.

Implementation Language:

The language we chose for our implementation is Java. We felt this was the most logical choice of languages, as this program would be hypothetically deployed on any number of operating systems and architectures. Since Java has a JVM for nearly all modern architectures, a Java client program would run anywhere it is required to.

This client program does not need the fast execution speed of a real time system, nor extreme high performance that is associated with languages like C++ or C, so java was a more than capable language in this respect.

For our server and database we would implement them in C++, not just for its performance but also because our servers would more than likely be running on architecturally similar machines. This would mean that we wouldn't need to make the sizable changes required to make our client code executable on several OS/architectures.

Architecture Diagram
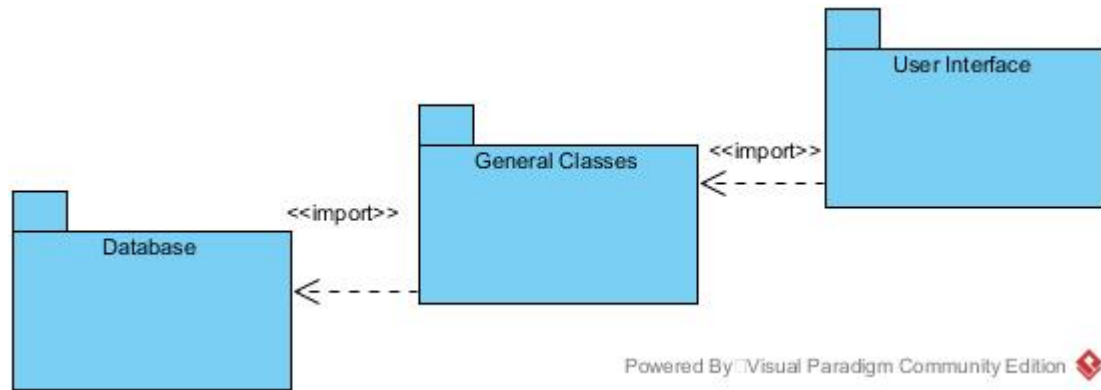
Analysis Artefacts

Identifying Candidate classes

Candidate Objects:

Using the data driven design method rather than responsibility design method we listed many potential candidate classes. We did this by utilizing the noun identification technique.

The initial list.

Username	Password	Screen	Form	Registration	Link
Email	Session	Game	System	Play	Solo
Party	Invite	Community	Group	Profile	User
Moderator	Notification	Request	Search	Window	Display
Tag	Database	Actor	Condition	Text	Data
Address	Friend	Decline	Accept	List	Message
Update	Send	Input	Dialog	Leader	

Heuristics:
1. Too vague or too specific – RED
2. Is it an attribute? – YELLOW
3. Is it an operation? – BLUE
4. Out of scope – GREEN
5. Equivalent (other similar objects) – PURPLE

The filtered list:

Party	Moderator	Session	Invite	Database
Friend	Game	Community	Form	Profile
Window	Leader	User	Message	

Party:

A party is a collection of Users that would be entering and playing a game together. They would be able to interact through a messaging system. Every party will have a party leader assigned to it who can remove other members.

Moderator:

A moderator is a user with extra privileges assigned to a community in order to monitor and respond to any issues between users who are part of the community. A moderator can remove users from a community and ban them from re-joining permanently.

Session:

A session would encompass and hold relevant details for the user while they are actively using the application. A session lasts from the time a user logs in to the application to the time they logout.

Invite:

There are three types of invites:
1. Party invite: This contains party id, and id of receiver. Used to get confirmation or refusal of addition to party roster. Notification is sent in response to choice.
2. Community invite: A request to join a community. Contains, community ID and id of sender. Received by community moderator. Used to get confirmation or refusal to gain access to a community group. Notification is sent in response to choice.
3. Friend invite: This contains sender id and receiver id. Received user who approves or declines the invitation. Notification is sent in response to choice.

Database:

The database provides storage for information such as; User details, Communities and their details, Messages, Outstanding Invites etc.

Friend:

A user who gives permission for another user to see their online status and allow to message them directly.

Game:

A game is the game of choice that a player wishes to launch and play. This can either be done alone or as part of a party.

Community:

A community is a large collection of users with similar interests. These communities can be created for a variety of interests that bring people together and allows them to interact.

Form:

A generalised term for a way to input information from the user. Primary uses include registration, login and messaging.

Profile:

A profile encompasses all the user's data that can be changed at any time and is public to all users. This includes a bio, favorite game, full name, date of birth, etc. Some of this information can be flagged as private.

Window:

This is the graphical entity that the user interacts with once they launch the application.

Leader:

A leader is the person assigned in a party that delegates which game is played. The only person allowed to remove players from the party. By default this user who creates a party is assigned that party's leader.

Message:

A general term for all communication between users. This includes personal messages and invites. A message would consists of the recipient's username, sender's username and text.

User:

A general term for anyone who launches and uses the application.

Class Diagram:
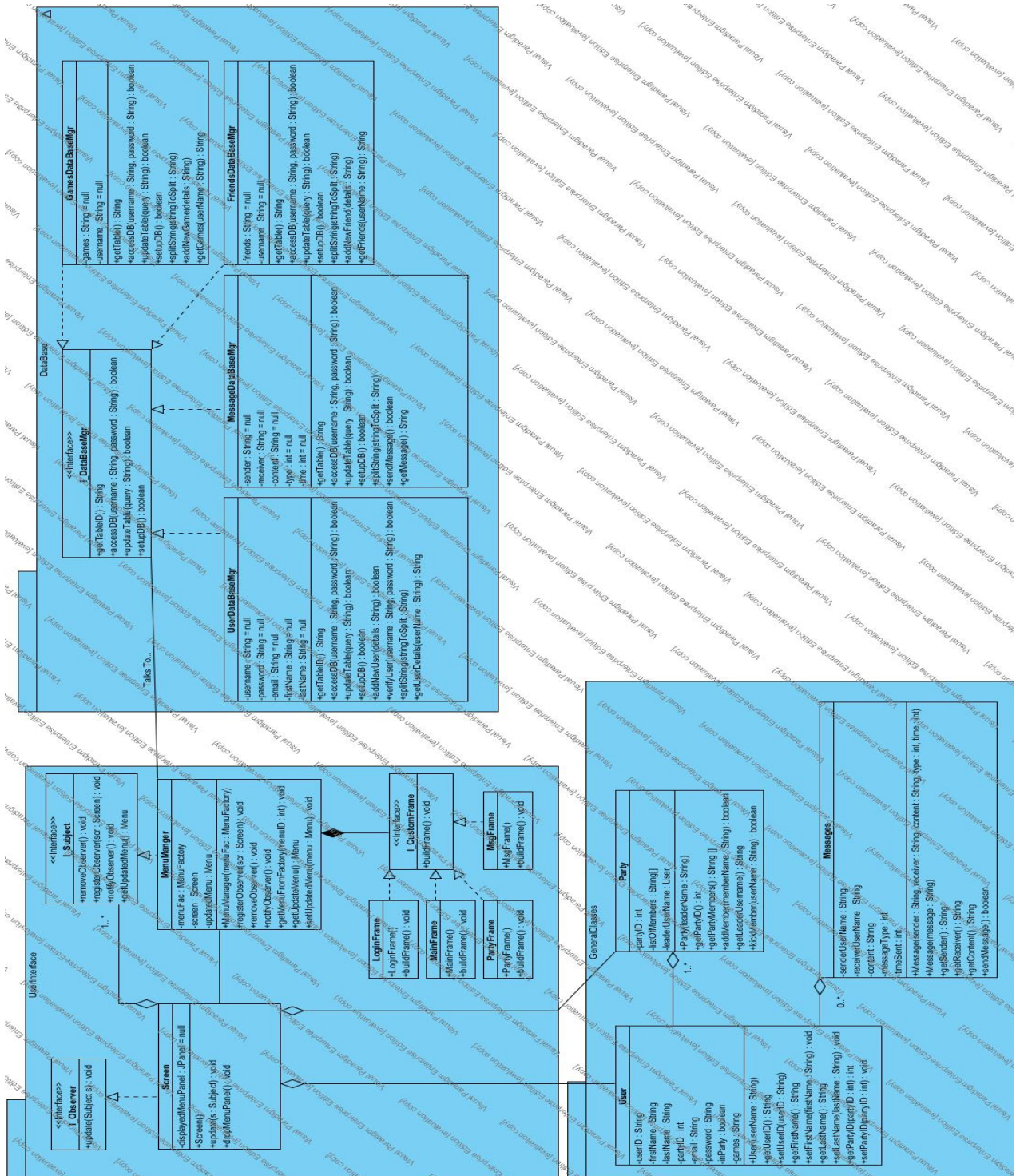
Interaction Diagram:
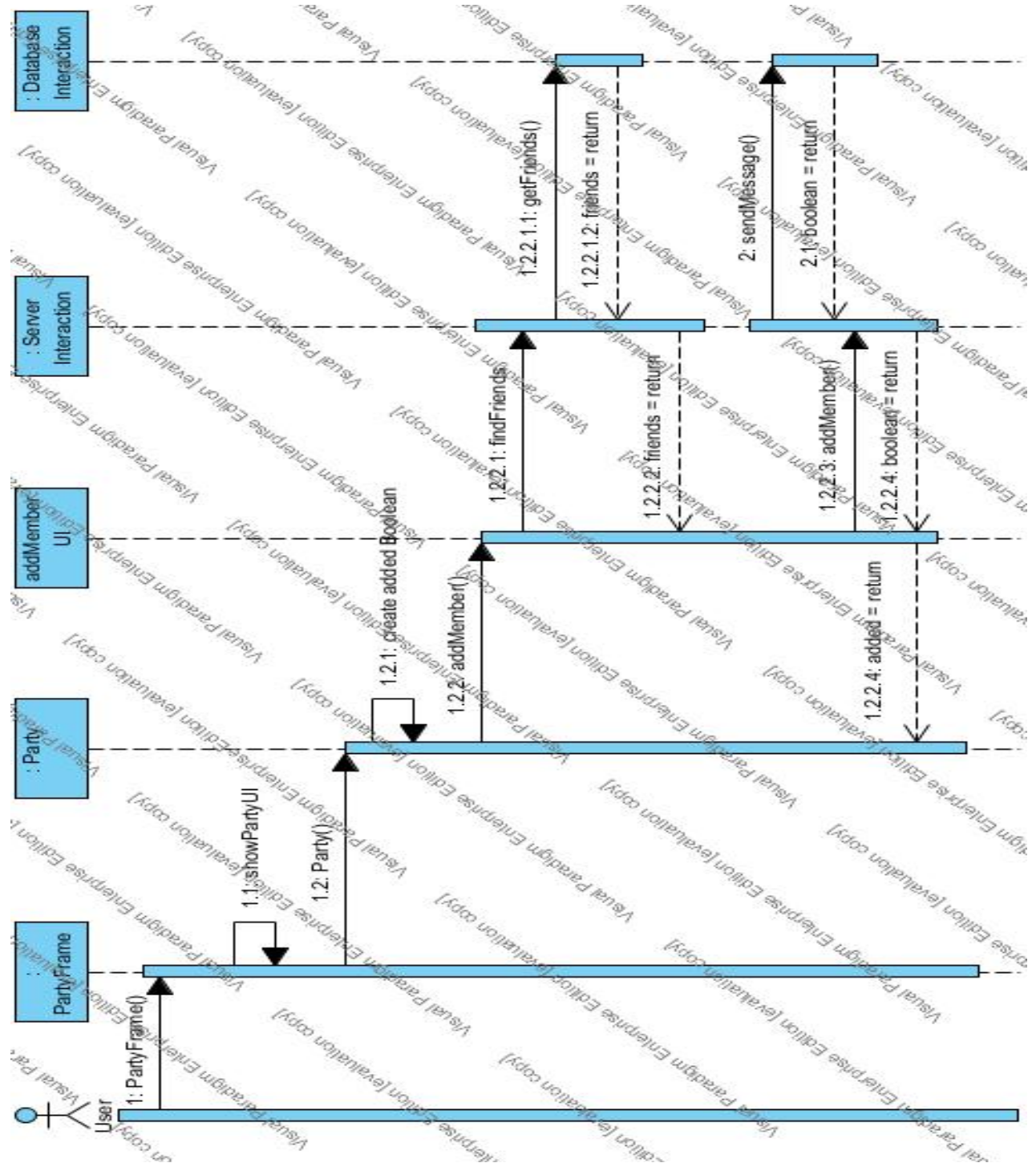
Entity Relationship Diagram:
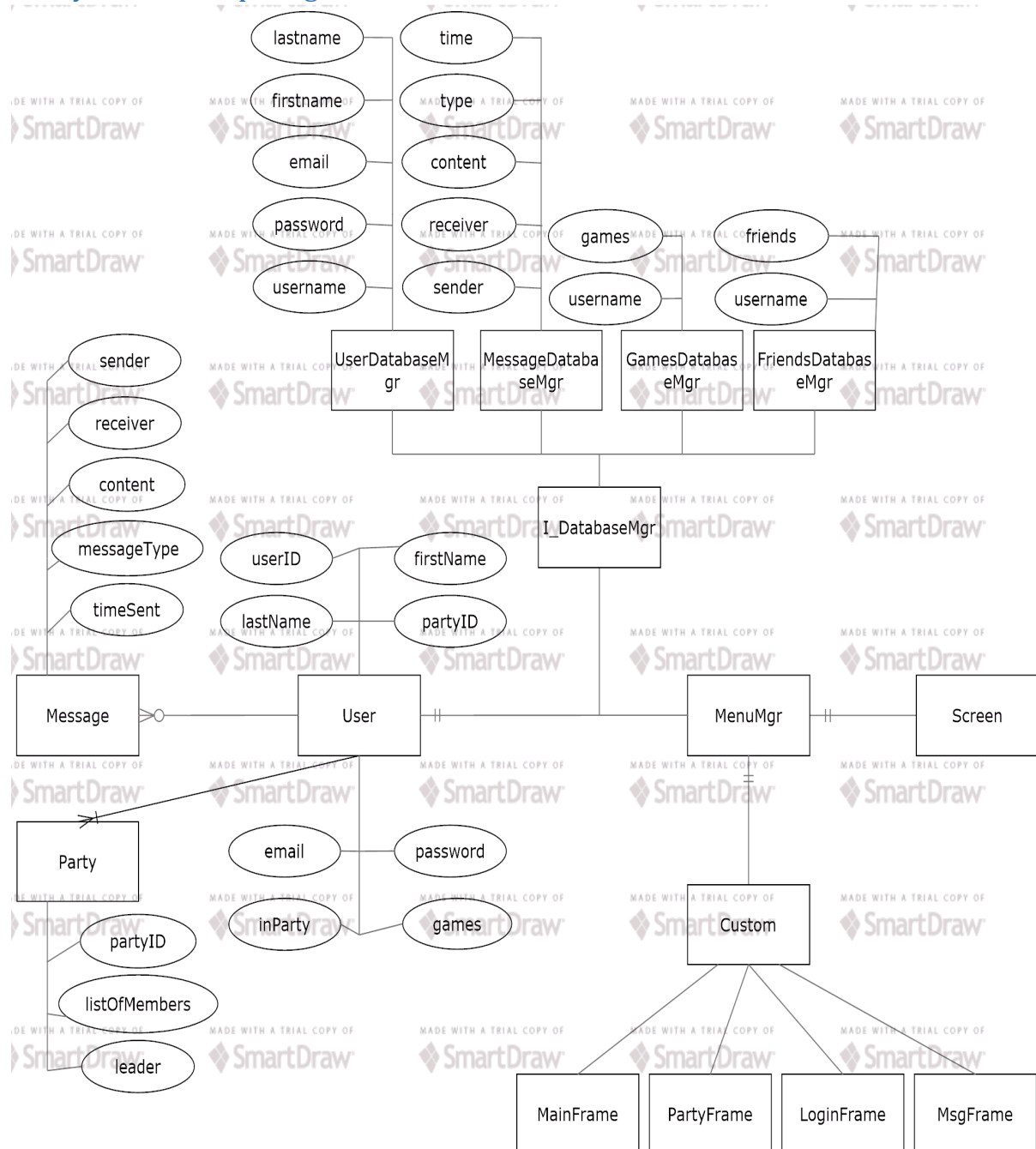
Code Implementation

Database Package:

DatabaseInterface.java

```java
/*
* Anon., in Collins, J. (ed.) (2015) 'CS4125: Serenity Gaming – a sample project from Sem1 2014-2015',
* CS4125: Systems Analysis & Design [online],
* available: http://www.csis.ul.ie/coursemodule/CS4125 [accessed 20 Oct 2015]
*
*  Adapted file: Serenity_Gaming/DatabaseInterface.java
*  Approximate use: 30%
**/

package database;

import java.util.ArrayList;

public interface DatabaseInterface {

    public abstract boolean canLogin(String username, String password) throws Exception;

    /* References the local Player class */
    public abstract String getPlayerDetails(String username) throws Exception;

    public abstract ArrayList<Integer> getPlayerFriendList(int playerID) throws Exception;

    public abstract ArrayList<Integer[]> getPlayerInvites(int playerID) throws Exception;

    public abstract void createPlayer(String username, String password, String email) throws Exception;

    public abstract int createParty(int partyLeaderID) throws Exception;

    public abstract ArrayList<Integer> getPartyDetails(int partyID, int playerID) throws Exception;

    public abstract boolean isPartyFull(int partyID) throws Exception;

    public abstract void addPlayerToParty(int playerID, int partyID) throws Exception;

    public abstract void removePlayerFromParty(int partyID, int playerID) throws Exception;

    public abstract int checkUserNameAndEmail(String username, String email) throws Exception;

    public abstract boolean doesPartyExist(int partyID) throws Exception;

    public abstract int doesPlayerExist(String username) throws Exception;

    public abstract boolean isPlayerInParty(int playerID) throws Exception;

    public abstract void addInvite(int senderID, int receiverID, int partyID) throws Exception;

    public abstract void removeInvite(int senderID, int receiverID, int partyID) throws Exception;
}
```

DatabaseAccess.java

```
/*
* Anon., in Collins, J. (ed.) (2015) 'CS4125: Serenity Gaming – a sample project from Sem1 2014-2015',
* CS4125: Systems Analysis & Design [online],
* available: http://www.csis.ul.ie/coursemodule/CS4125 [accessed 20 Oct 2015]
*
*   Adapted file: Serenity_Gaming/DatabaseAccess.java
*   Approximate use: 5%
**/
package database;

import java.io.File;
import java.io.FileWriter;
import java.io.PrintWriter;
import java.util.ArrayList;
import java.util.Scanner;

public class DatabaseAccess implements DatabaseInterface {

    private PrintWriter pWriter;
    private FileWriter fWriter;
    private File file;
    private Scanner fileReader;
    private final String USER_DETAILS_FILE = "UserDetails.txt";
    private final String FRIEND_DETAILS_FILE = "FriendDetails.txt";
    private final String PARTY_DETAILS_FILE = "PartyDetails.txt";
    private final String INVITE_DETAILS_FILE = "InviteDetails.txt";

    public DatabaseAccess() throws Exception {

    }

    @Override
    public boolean canLogin(String username, String password) throws Exception {
        file = new File(USER_DETAILS_FILE);
        if (file.exists()) {
            fileReader = new Scanner(file);
            while (fileReader.hasNextLine()) {
                String[] lineFromFile = (fileReader.nextLine()).split(",");
                if (lineFromFile[2].equals(password)) {
                    fileReader.close();
                    file = null;
                    return true;
                }
            }
        }

        fileReader.close();
        file = null;
        return false;
    }

    @Override
    public String getPlayerDetails(String username) throws Exception {
        file = new File(USER_DETAILS_FILE);
```

```java
      fileReader = new Scanner(file);
      String playerDetails = "";
      while (fileReader.hasNextLine()) {
         String[] lineFromFile = (fileReader.nextLine()).split(",");
         if (lineFromFile[1].equals(username)) {
            playerDetails += lineFromFile[0] + ",";
            playerDetails += lineFromFile[1] + ",";
            playerDetails += lineFromFile[3] + ",";
         }
      }
      fileReader.close();
      file = null;
      return playerDetails;
   }

   @Override
   public ArrayList<Integer> getPlayerFriendList(int playerID) throws Exception {
      String pidStr = playerID + "";
      ArrayList<Integer> friendIDs = new ArrayList<Integer>();
      file = new File(FRIEND_DETAILS_FILE);
      fileReader = new Scanner(file);
      while (fileReader.hasNextLine()) {
         String lineFromFile[] = (fileReader.nextLine()).split(",");
         if (pidStr.equals(lineFromFile[0])) {
            friendIDs.add(Integer.parseInt(lineFromFile[1]));
         } else if (pidStr.equals(lineFromFile[1])){
            friendIDs.add(Integer.parseInt(lineFromFile[0]));
         }
      }
      fileReader.close();
      file = null;

      return friendIDs;
   }

   @Override
   public ArrayList<Integer[]> getPlayerInvites(int playerID) throws Exception {
      ArrayList<Integer[]> idList = new ArrayList<Integer[]>();
      Integer [] ids = new Integer [2];
      file = new File(INVITE_DETAILS_FILE);
      fileReader = new Scanner(file);
      while (fileReader.hasNextLine()) {
         String[] lineFromFile = (fileReader.nextLine()).split(",");
         if (playerID == Integer.parseInt(lineFromFile[1])) {
            ids[0] = Integer.parseInt(lineFromFile[0]);
            ids[1] = Integer.parseInt(lineFromFile[2]);
            if (!idList.contains(ids)) {
               idList.add(ids);
            }
         }
      }
      fileReader.close();
      file = null;
      return idList;
   }

   @Override
```

```java
public void createPlayer(String username, String password, String email) throws Exception {
    file = new File(USER_DETAILS_FILE);
    ArrayList<String> userDetails = fileToList();
    String playerDetails = "";

    if (userDetails.size() > 0) {
        String[] lastEntry = (userDetails.get(userDetails.size() - 1)).split(",");
        int newPlayerID = Integer.parseInt(lastEntry[0]) + 1;
        playerDetails = newPlayerID + "," + username + "," + password + "," + email;
    } else {
        playerDetails += 1 + "," + username + "," + password + "," + email;
    }
    fWriter = new FileWriter(file, true);
    pWriter = new PrintWriter(fWriter);
    pWriter.println(playerDetails);
    pWriter.close();
    fWriter.close();
    file = null;
}

@Override
public int createParty(int partyLeaderID) throws Exception {
    file = new File(PARTY_DETAILS_FILE);
    ArrayList<String> parties = fileToList();
    String newParty = "";
    int newPartyID = 1;
    if(parties.size() > 0){
                    String [] lastEntry = (parties.get(parties.size() - 1)).split(",");
                    newPartyID = Integer.parseInt(lastEntry[0]) + 1;
                    newParty += newPartyID + "," + partyLeaderID;
        }
        else{
            newParty += 1 + "," +  partyLeaderID;
        }

    fWriter = new FileWriter(file, true);
    pWriter = new PrintWriter(fWriter);
    pWriter.println(newParty);
    pWriter.close();
    fWriter.close();
    file = null;

    return newPartyID;
}

@Override
public ArrayList<Integer> getPartyDetails(int partyID, int playerID) throws Exception {
    file = new File(PARTY_DETAILS_FILE);
    fileReader = new Scanner(file);
    ArrayList<Integer> partyDetails = new ArrayList<Integer>();
    while (fileReader.hasNextLine()) {
        String lineFromFile[] = fileReader.nextLine().split(",");
        if (partyID == Integer.parseInt(lineFromFile[0])) {
            for (String x : lineFromFile) {
                partyDetails.add(Integer.parseInt(x));
            }
```

```java
        }
      }
      fileReader.close();
      file = null;
      if (!partyDetails.contains(playerID)) {
        partyDetails.clear();
      }
      return partyDetails;
    }

    @Override
    public boolean isPartyFull(int partyID) throws Exception {
      boolean partyFull = true;
      file = new File(PARTY_DETAILS_FILE);
      fileReader = new Scanner(file);
      while (fileReader.hasNextLine()) {
        String lineFromFile[] = fileReader.nextLine().split(",");
        if (partyID == Integer.parseInt(lineFromFile[0])) {
          if (lineFromFile.length < 6) {
            partyFull = false;
            break;
          }
        }
      }
      fileReader.close();
      file = null;
      return partyFull;
    }

    @Override
    public void addPlayerToParty(int playerID, int partyID) throws Exception {
      file = new File(PARTY_DETAILS_FILE);
      ArrayList<String> parties = fileToList();

      for (int i = 0; i < parties.size(); i++) {
        String[] lineFromFile = parties.get(i).split(",");
        if (partyID == Integer.parseInt(lineFromFile[0])) {
          String amendedParty = parties.get(i);
          amendedParty += "," + playerID;
          parties.remove(i);
          parties.add(i, amendedParty);
          break;
        }
      }
      writeToFile(parties, false);
      file = null;
    }

    @Override
    public void removePlayerFromParty(int partyID, int playerID) throws Exception {
      file = new File(PARTY_DETAILS_FILE);
      ArrayList<String> parties = fileToList();

      for (int i = 0; i < parties.size(); i++) {
        String amendedParty = "";
        String[] partyInfo = (parties.get(i)).split(",");
        if (partyID == Integer.parseInt(partyInfo[0])) {
```

```java
            amendedParty += partyInfo[0];
            for (int j = 1; j < partyInfo.length; j++) {
               if (playerID != Integer.parseInt(partyInfo[j])) {
                  amendedParty += "," + partyInfo[j];
               }
            }
            parties.remove(i);
            if (amendedParty.contains(",")) {
               parties.add(i, amendedParty);
            }
            break;
         }
      }
      writeToFile(parties, false);
      file = null;
   }

   @Override
   public int checkUserNameAndEmail(String username, String email) throws Exception {
      file = new File(USER_DETAILS_FILE);
      fileReader = new Scanner(file);

      while (fileReader.hasNextLine()) {
         String[] lineFromFile = (fileReader.nextLine()).split(",");
         if (lineFromFile[1].equals(username)) {
            fileReader.close();
            file = null;
            return 0;
         } else if (lineFromFile[3].equals(email)) {
            fileReader.close();
            file = null;
            return 1;
         }
      }
      fileReader.close();
      file = null;
      return 2;
   }

   @Override
   public boolean doesPartyExist(int partyID) throws Exception {
      file = new File(PARTY_DETAILS_FILE);
      fileReader = new Scanner(file);
      boolean partyExists = false;
      while (fileReader.hasNextLine()) {
         String[] lineFromFile = fileReader.nextLine().split(",");
         if (partyID == Integer.parseInt(lineFromFile[0]));
         {
            partyExists = true;
         }
      }
      fileReader.close();
      file = null;
      return partyExists;
   }

   @Override
```

```java
public int doesPlayerExist(String username) throws Exception {
    file = new File(USER_DETAILS_FILE);
    fileReader = new Scanner(file);
    while (fileReader.hasNextLine()) {
        String[] lineFromFile = fileReader.nextLine().split(",");
        if (username.equals(lineFromFile[1])) {
            fileReader.close();
            file = null;
            return Integer.parseInt(lineFromFile[0]);
        }
    }
    fileReader.close();
    file = null;
    return 0;
}

@Override
public boolean isPlayerInParty(int playerID) throws Exception {
    boolean isInParty = false;
    file = new File(PARTY_DETAILS_FILE);
    fileReader = new Scanner(file);
    while (fileReader.hasNextLine()) {
        String[] lineFromFile = fileReader.nextLine().split(",");
        for (int i = 1; i < lineFromFile.length; i++) {
            if (playerID == Integer.parseInt(lineFromFile[i])) {
                isInParty = true;
                break;
            }
        }
    }
    fileReader.close();
    file = null;

    return isInParty;
}

@Override
public void addInvite(int senderID, int receiverID, int partyID) throws Exception {
    file = new File(INVITE_DETAILS_FILE);
    String newInvite = "" + senderID + "," + receiverID + "," + partyID;

    fWriter = new FileWriter(file, true);
    pWriter = new PrintWriter(fWriter);
    pWriter.println(newInvite);
    pWriter.close();
    fWriter.close();
    file = null;
}

@Override
public void removeInvite(int senderID, int receiverID, int partyID) throws Exception {
    file = new File(INVITE_DETAILS_FILE);
    ArrayList<String> invites = fileToList();

    for (int i = 0; i < invites.size(); i++) {
        String[] lineFromFile = invites.get(i).split(",");
        if (senderID == Integer.parseInt(lineFromFile[0]) && receiverID == Integer.parseInt(lineFromFile[1])
```

```
                && partyID == Integer.parseInt(lineFromFile[2])) {
            invites.remove(i);
            break;
          }
      }
      writeToFile(invites, false);
      file = null;
  }

  private ArrayList<String> fileToList() {
      ArrayList<String> list = new ArrayList<String>();
      try {
          fileReader = new Scanner(file);
          while (fileReader.hasNextLine()) {
            list.add(fileReader.nextLine());
          }
          fileReader.close();
      } catch (Exception e) {
      }
      return list;
  }

  private void writeToFile(ArrayList<String> list, boolean amended) {
      try {
          fWriter = new FileWriter(file, amended);
          pWriter = new PrintWriter(fWriter);
          for (int i = 0; i < list.size(); i++) {
            pWriter.println(list.get(i));
          }

          pWriter.close();
          fWriter.close();
      } catch (Exception e) {
      }
  }
}
```

Session Package

Collection.java

```
package session;

import java.util.ArrayList;

public interface Collection<E> {

  public abstract void add(E item);

  public abstract E get(int id);

  public abstract ArrayList<E> getAll();

  public abstract void remove(int id, int pid);
}
```

Invite.java

```java
package session;

public abstract class Invite
{
    protected int senderID;
    protected int receiverID;
    protected int partyID;
    protected String message;

    public abstract int getSenderID();

    public abstract int getReceiverID();

    public abstract int getPartyID();

    public abstract String getMessage();

    public abstract boolean equals(Invite otherInvite);
}
```

InviteCollection.java

```java
package session;

import java.util.ArrayList;

public class InviteCollection implements Collection<Invite> {

    private ArrayList<Invite> invites;

    public InviteCollection() {
        invites = new ArrayList<Invite>();
    }

    @Override
    public void add(Invite item) {
        if (invites.isEmpty()) {
            invites.add(item);
        } else {
            boolean newItem = true;
            for (int x =0; x < invites.size(); x++) {
                if (item.equals(x)) {
                    newItem = false;
                    break;
                }
                else if(invites.get(x).getSenderID() == item.getSenderID())
                {
                    invites.remove(x);
                    x--;
                }
            }
            if (newItem) {
                invites.add(item);
            }
        }
    }
```

```java
    @Override
    public  Invite get(int id)
    {
        //Invite inv = new PartyInvite();
        for(Invite x : invites)
        {
            if(id == x.getSenderID())
            {
                return x;
            }
        }
        return null;
    }

    @Override
    public ArrayList<Invite> getAll()
    {
        return invites;
    }

    @Override
    public void remove(int id, int pid)
    {
        for(int x = 0; x < invites.size(); x++)
        {
            if(id == invites.get(x).getSenderID() && pid == invites.get(x).getPartyID())
            {
                invites.remove(x);
                x--;
            }
        }
    }
}
```

InviteFactory.java

```java
package session;

public abstract class InviteFactory
{
    public abstract Invite createInvite(int senderID, int receiverID, int partyID);
}
```

Party.java

```java
package session;

import java.util.ArrayList;

public class Party implements SessionObserver {

    private int id;
    private int leaderID;
    private ArrayList<Integer> partyMembers;

    public Party() {
        partyMembers = new ArrayList<Integer>();
```

```java
    }

    @Override
    public void update(SessionSubject s) {
        ArrayList<Integer> partyInfo = s.getState();
        if (!partyInfo.isEmpty()) {
            id = partyInfo.get(0);
            partyMembers.clear();
            for (int i = 1; i < partyInfo.size(); i++) {
                partyMembers.add(partyInfo.get(i));
            }
            leaderID = partyMembers.get(0);
        } else {
            id = 0;
            leaderID = 0;
            partyMembers.clear();
        }
    }

    public boolean isPartyLeader(int id) {
        return (leaderID == id);
    }

    public boolean doesPartyExist() {
        if (partyMembers.size() == 0) {
            return false;
        }
        return true;
    }

    public int getId() {
        return id;
    }

    public int getPartySize() {
        return (partyMembers.size() > 0) ? (partyMembers.size()) : 0;
    }
    public ArrayList<Integer> getPartyMembers()
    {
        return partyMembers;
    }
    public boolean isMember(int id)
    {
        boolean validMember = false;
        if(partyMembers.size() > 1 && partyMembers.contains(id))
        {
            validMember = true;
        }
        return validMember;
    }
}
```

PartyInvite.java

```java
package session;

public class PartyInvite extends Invite
```

```java
{
  public PartyInvite(int senderID, int receiverID, int partyID)
  {
    this.senderID = senderID;
    this.receiverID = receiverID;
    this.partyID = partyID;
  }
  @Override
  public int getSenderID() {
    return senderID;
  }

  @Override
  public int getReceiverID() {
    return receiverID;
  }

  @Override
  public String getMessage() {
    return "Party Invite received from User " + senderID;
  }

  @Override
  public boolean equals(Invite otherInvite) {
    return (senderID == otherInvite.getSenderID() && partyID == otherInvite.partyID);
  }

  @Override
  public int getPartyID() {
    return partyID;
  }
}
```

PartyInviteFactory.java
```java
package session;

public class PartyInviteFactory extends InviteFactory {

  @Override
  public Invite createInvite(int senderID, int receiverID, int partyID)
  {
    return new PartyInvite(senderID, receiverID, partyID);
  }
}
```

Player.java
```java
/*
* Anon., in Collins, J. (ed.) (2015) 'CS4125: Serenity Gaming – a sample project from Sem1 2014-2015',
* CS4125: Systems Analysis & Design [online],
* available: http://www.csis.ul.ie/coursemodule/CS4125 [accessed 20 Oct 2015]
*
*  Adapted file: Serenity_Gaming/Player.java
*  Approximate use: 20%
**/
package session;
```

```java
import java.util.ArrayList;

public class Player extends User implements SessionSubject {

    private static Player player = null;
    ArrayList<Integer> friends;
    ArrayList<Integer> partyInformation;
    ArrayList<SessionObserver> observers;
    Collection inviteCollection;
    private String email;

    private Player() {
        observers = new ArrayList<SessionObserver>();
        friends = new ArrayList<Integer>();
        partyInformation = new ArrayList<Integer>();
        inviteCollection = new InviteCollection();
    }

    public static Player getInstance() {
        if (player == null) {
            player = new Player();
        }
        return player;
    }

    @Override
    public void attach(SessionObserver o) {
        observers.add(o);
    }

    @Override
    public void detach(SessionObserver o) {
        if (observers.contains(o)) {
            observers.remove(o);
        }
    }

    @Override
    public void nofity() {
        if (!observers.isEmpty()) {
            for (SessionObserver observer : observers) {
                observer.update(this);
            }
        }
    }

    public void resetValues() {
        id = 0;
        name = "";
        email = "";
    }

    @Override
    public int getId() {
        return id;
    }
```

```java
@Override
public void setId(int id) {
    this.id = id;
}

@Override
public String getName() {
    return name;
}

@Override
public void setName(String name) {
    this.name = name;
}

public String getEmail() {
    return email;
}

public void setEmail(String email) {
    this.email = email;
}

public void addFriend(int friendID) {
    friends.add(friendID);
}

public void addInvite(Invite invite) {
    inviteCollection.add(invite);
}

public void removeInvite(int senderID, int partyID) {
    inviteCollection.remove(senderID, partyID);
}

public ArrayList<Invite> getInvites() {
    return inviteCollection.getAll();
}
@Override
public ArrayList<Integer> getState() {
    return partyInformation;
}

public void addToPartyInformation(int info)
{
    partyInformation.add(info);
}

public void updatePartyInformation(ArrayList<Integer> info) {
    partyInformation = info;
}

public void clearPartyInformation() {
    partyInformation.clear();
}
```

```java
      boolean isFriend(int id) {
         return friends.contains(id);
      }
   }
}
```

SessionInformation.java

```java
/*
 * Anon., in Collins, J. (ed.) (2015) 'CS4125: Serenity Gaming – a sample project from Sem1 2014-2015',
 * CS4125: Systems Analysis & Design [online],
 * available: http://www.csis.ul.ie/coursemodule/CS4125 [accessed 20 Oct 2015]
 *
 *  Adapted file: Serenity_Gaming/ProcessInput.java
 *  Approximate use: 40%
**/
package session;

import database.DatabaseInterface;
import java.util.ArrayList;

public class SessionInformation {

   private static SessionInformation sessionInfo = null;
   private DatabaseInterface database;
   private InviteFactory inviteFactory;
   private Player player = null;
   private Party party;

   private SessionInformation() {
      setPlayer();
      party = new Party();
      player.attach(party);
      inviteFactory = new PartyInviteFactory();
   }

   public static SessionInformation getInstance() {
      if (sessionInfo == null) {
         sessionInfo = new SessionInformation();
      }

      return sessionInfo;
   }

   public void setPlayer() {
      player = Player.getInstance();
   }

   public void setDbConnection(DatabaseInterface database) {
      this.database = database;
   }

   public boolean canUserLogin(String username, String password) {
      boolean canLogin = false;

      try {
         canLogin = database.canLogin(username, password);
         if (canLogin) {
```

```java
            getPlayerDetails(username);
            getPlayerFriendList();
            getPlayerInvites();
        }
    } catch (Exception e) {
        System.out.println("Error logging in: " + e.toString());
    }

    return canLogin;
}

public boolean doesPlayerExist(String username) {
    boolean exists = false;

    try {
        if (database.doesPlayerExist(username) != 0) {
            exists = true;
        }
    } catch (Exception e) {
        System.out.println("doesPlayerExist Error: " + e.toString());
    }

    return exists;
}

public int checkUsernameEmail(String username, String email) {
    int existsType = -1;

    try {
        existsType = database.checkUserNameAndEmail(username, email);
    } catch (Exception e) {
        System.out.println("Error validaing username/email: " + e.toString());
    }

    return existsType;
}

public void createPlayer(String username, String password, String email) {
    try {
        database.createPlayer(username, password, email);
    } catch (Exception e) {
        System.out.println("Error creating player: " + e.toString());
    }
}

public void getPlayerDetails(String username) {
    try {
        String[] details = database.getPlayerDetails(username).split(",");

        this.player.setId(Integer.parseInt(details[0]));
        this.player.setName(details[1]);
        this.player.setEmail(details[2]);
    } catch (Exception e) {
        System.out.println("Error getting player details: " + e.toString());
    }
}
```

```java
public void getPlayerFriendList() {
    try {
        ArrayList<Integer> friends = database.getPlayerFriendList(player.getId());
        if (friends.size() > 0) {
            for (int i = 0; i < friends.size(); i++) {
                player.addFriend(friends.get(i));
            }
        }
    } catch (Exception e) {
    }
}

public void getPlayerInvites() {
    try {
        ArrayList<Integer[]> invites = database.getPlayerInvites(player.getId());
        if (invites.size() > 0) {
            for (int i = 0; i < invites.size(); i++) {
                Invite newInvite = inviteFactory.createInvite(invites.get(i)[0], player.getId(), invites.get(i)[1]);
                player.addInvite(newInvite);
            }
        }
    } catch (Exception e) {
    }
}

public void getPartyDetails() {
    try {
        ArrayList<Integer> partyDetails = database.getPartyDetails(party.getId(), player.getId());

        player.clearPartyInformation();
        player.updatePartyInformation(partyDetails);
        player.nofity();
    } catch (Exception e) {
    }
}

public boolean isPlayerInParty() {
    return party.doesPartyExist();
}

public boolean isMemberOfParty(int id) {
    return party.isMember(id);
}

public boolean isFriend(int id) {
    return player.isFriend(id);
}

public void createParty() {
    try {
        int partyID = database.createParty(player.getId());
        player.addToPartyInformation(partyID);
        player.addToPartyInformation(player.getId());
        player.nofity();
    } catch (Exception e) {

    }
```

```java
    }

    public void leaveParty() {
        try {
            int partyID = party.getId();
            System.out.println(partyID);
            System.out.println(player.getId());
            database.removePlayerFromParty(partyID, player.getId());
            player.clearPartyInformation();
            player.nofity();
        } catch (Exception e) {
        }
    }

    public String getPlayerName() {
        return player.getName();
    }

    public void logPlayerOut() {
        if (party.doesPartyExist()) {
            leaveParty();
        }
        System.out.println("logged out");
        player.resetValues();
    }

    public boolean isPartyLeader() {
        return party.isPartyLeader(player.getId());
    }

    public int getPartySize() {
        return party.getPartySize();
    }

    public void addPlayerToParty(int partyID) {
        try {
            player.clearPartyInformation();
            player.addToPartyInformation(partyID);
            player.addToPartyInformation(player.getId());
            player.nofity();
            database.addPlayerToParty(player.getId(), partyID);

            getPartyDetails();
        } catch (Exception e) {
        }
    }

    public void removePlayerFromParty(int playerID) {
        try {
            database.removePlayerFromParty(party.getId(), playerID);
        } catch (Exception e) {
        }
    }

    public ArrayList<String> getInviteMessages() {
        ArrayList<String> invMsg = new ArrayList<String>();
        ArrayList<Invite> invites = player.getInvites();
```

```java
      for (int i = 0; i < invites.size(); i++) {
         invMsg.add(invites.get(i).getMessage());
      }

      return invMsg;
   }

   public ArrayList<Integer> getPartyMembers() {
      return party.getPartyMembers();
   }

   public void sendInvite(int friendToInvite) {
      try {
         database.addInvite(player.getId(), friendToInvite, party.getId());
      } catch (Exception ex) {
      }
   }

   public int getPartyIDFromSenderInvite(int playerID)
   {
      ArrayList<Invite> myInvites = player.getInvites();
      int partyID = 0;
      for (int i = 0; i < myInvites.size(); i++) {
         if (playerID == myInvites.get(i).getSenderID()) {
            partyID = myInvites.get(i).getPartyID();
            break;
         }
      }
      System.out.println(partyID);
      return partyID;
   }
   public void removeInvite(int playerID) {
      int partyID = getPartyIDFromSenderInvite(playerID);
      player.removeInvite(playerID, partyID);
      try {
         database.removeInvite(playerID, player.getId(), partyID);
      } catch (Exception e) {
         System.out.println("check 4");
      }
   }
   public int getPlayerId()
   {
      return player.getId();
   }
}
```

SessionObserver.java
```java
package session;

public interface SessionObserver
{
   public abstract void update(SessionSubject s);
}
```

SessionSubject.java
```java
package session;
```

```java
import java.util.ArrayList;

public interface SessionSubject
{
    public abstract void attach(SessionObserver o);

    public abstract void detach(SessionObserver o);

    public abstract void nofity();

    public abstract ArrayList<Integer> getState();
}
```

User.java

```java
package session;

public abstract class User {
    protected int id;
    protected String name;

    public abstract int getId();

    public abstract void setId(int id);

    public abstract String getName();

    public abstract void setName(String name);
}
```

Userinterface Package

MainMenuUI.java

```java
/*
* Anon., in Collins, J. (ed.) (2015) 'CS4125: Serenity Gaming – a sample project from Sem1 2014-2015',
* CS4125: Systems Analysis & Design [online],
* available: http://www.csis.ul.ie/coursemodule/CS4125 [accessed 20 Oct 2015]
*
*   Adapted file: Serenity_Gaming/MenuUI.java
*   Approximate use: 70%
**/
package userinterface;

import java.awt.BorderLayout;
import java.awt.Color;
import java.awt.FlowLayout;
import java.awt.GridLayout;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import javax.swing.JButton;
import javax.swing.JLabel;
import javax.swing.JOptionPane;
import javax.swing.JPanel;

public class MainMenuUI extends Menu {
```

```java
public MainMenuUI() {
    showMainMenu();
}

public void showMainMenu() {
    JPanel mainMenuPanel = new JPanel();
    BorderLayout mainMenuLayout = new BorderLayout();
    mainMenuPanel.setLayout(mainMenuLayout);

    JPanel topBarPanel = new JPanel();
    FlowLayout topBarLayout = new FlowLayout();
    topBarLayout.setAlignment(FlowLayout.RIGHT);
    topBarPanel.setLayout(topBarLayout);
    JLabel welcomeLabel = new JLabel("Welcome ");
    topBarPanel.add(welcomeLabel);
    JLabel spacer = new JLabel("        ");
    topBarPanel.add(spacer);
    JButton logoutButton = new JButton("Logout");
    logoutButton.addActionListener(new ActionListener() {
        @Override
        public void actionPerformed(ActionEvent e) {

            sessionInfo.logPlayerOut();
            menuMgr.getMenuFromFactory(1);

        }
    });
    topBarPanel.add(logoutButton);
    mainMenuPanel.add(topBarPanel, mainMenuLayout.NORTH);

    JPanel centerMenuPanel = new JPanel();
    BorderLayout centerMenuLayout = new BorderLayout();
    centerMenuPanel.setLayout(centerMenuLayout);
    centerMenuPanel.setBackground(Color.blue);

    JPanel centerMenuButtonsPanel = new JPanel();
    GridLayout centerMenuButtonsLayout = new GridLayout(3, 3);
    centerMenuButtonsPanel.setLayout(centerMenuButtonsLayout);
    JButton gameButton = new JButton("Games");
    gameButton.addActionListener(new ActionListener() {
        @Override
        public void actionPerformed(ActionEvent e) {
            JOptionPane.showMessageDialog(null, "Component not integrated");
        }
    });
    centerMenuButtonsPanel.add(gameButton);
    JButton profileButton = new JButton("Profile");
    profileButton.addActionListener(new ActionListener() {
        @Override
        public void actionPerformed(ActionEvent e) {
            JOptionPane.showMessageDialog(null, "Component not integrated");
        }
    });
    centerMenuButtonsPanel.add(profileButton);
    JButton friendsButton = new JButton("Friends List");
    friendsButton.addActionListener(new ActionListener() {
        @Override
```

```java
      public void actionPerformed(ActionEvent e) {
        JOptionPane.showMessageDialog(null, "Component not integrated");
      }
    });
    centerMenuButtonsPanel.add(friendsButton);
    JButton communityButton = new JButton("Communities");
    communityButton.addActionListener(new ActionListener() {
      @Override
      public void actionPerformed(ActionEvent e) {
        JOptionPane.showMessageDialog(null, "Component not integrated");
      }
    });
    centerMenuButtonsPanel.add(communityButton);
    JButton partyButton = new JButton("Party");
    partyButton.addActionListener(new ActionListener() {
      @Override
      public void actionPerformed(ActionEvent e) {
        if (sessionInfo.isPlayerInParty()) {
          System.out.println("getting partyDetails");
          sessionInfo.getPartyDetails();

          menuMgr.getMenuFromFactory(3);
        } else {
          int choice = JOptionPane.showConfirmDialog(null, "You are Currently not a member of a
party.\nWould you like to create a new party?", "Create a Party", JOptionPane.YES_NO_OPTION);
          if (choice == JOptionPane.YES_OPTION) {
            System.out.println("clicked yes");
            sessionInfo.createParty();
          }
        }
      }
    });
    centerMenuButtonsPanel.add(partyButton);
    centerMenuButtonsPanel.add(communityButton);
    JButton messageButton = new JButton("Messages");
    messageButton.addActionListener(new ActionListener() {
      @Override
      public void actionPerformed(ActionEvent e) {
        sessionInfo.getPlayerInvites();
        menuMgr.getMenuFromFactory(4);
      }
    });
    centerMenuButtonsPanel.add(messageButton);
    centerMenuPanel.add(centerMenuButtonsPanel, centerMenuLayout.CENTER);
    centerMenuPanel.add(centerMenuButtonsPanel, centerMenuLayout.CENTER);

    mainMenuPanel.add(centerMenuPanel, mainMenuLayout.CENTER);
    panel = mainMenuPanel;
  }

  @Override
  public JPanel getMenuPanel() {
    return panel;
  }

  @Override
  public void setMenuManager(MenuManager menuMgr) {
```

```
      this.menuMgr = menuMgr;
   }
}
```

Menu.java

```
/*
* Anon., in Collins, J. (ed.) (2015) 'CS4125: Serenity Gaming – a sample project from Sem1 2014-2015',
* CS4125: Systems Analysis & Design [online],
* available: http://www.csis.ul.ie/coursemodule/CS4125 [accessed 20 Oct 2015]
*
*   Adapted file: Serenity_Gaming/Panel.java
*   Approximate use: 80%
**/
package userinterface;

import javax.swing.JPanel;
import session.SessionInformation;


public abstract class Menu {

   protected MenuManager menuMgr;
   protected SessionInformation sessionInfo;
   protected JPanel panel;

   public Menu() {

   }

   public abstract JPanel getMenuPanel();

   public void setMenuManager(MenuManager menuMgr) {
      this.menuMgr = menuMgr;
   }

   public void setSessionInformation(SessionInformation sessionInfo) {
      this.sessionInfo = sessionInfo;
   }
}
```

MenuFactory.java

```
/*
* Anon., in Collins, J. (ed.) (2015) 'CS4125: Serenity Gaming – a sample project from Sem1 2014-2015',
* CS4125: Systems Analysis & Design [online],
* available: http://www.csis.ul.ie/coursemodule/CS4125 [accessed 20 Oct 2015]
*
*   Adapted file: Serenity_Gaming/
*   Approximate use: 70%
**/
package userinterface;

import session.SessionInformation;

public class MenuFactory {

   private SessionInformation sessionInfo = null;
```

```java
    public MenuFactory() {
    }

    public MenuFactory(SessionInformation sessionInfo) {
        this.sessionInfo = sessionInfo;
    }

    public Menu getMenu(int menuID, MenuManager menuMgr) {
        Menu menu = null;

        switch (menuID) {
            case 1:
                menu = new StartScreenUI(500, 500);
                menu.setMenuManager(menuMgr);
                break;
            case 2:
                menu = new MainMenuUI();
                menu.setMenuManager(menuMgr);
                break;
            case 3:
                menu = new PartyUI();
                menu.setMenuManager(menuMgr);
                break;
            case 4:
                menu = new MessageUI();
                menu.setMenuManager(menuMgr);
                break;
        }
        menu.setSessionInformation(sessionInfo);
        menu.setMenuManager(menuMgr);
        return menu;
    }
}
```

MenuManager.java

```java
/*
 * Anon., in Collins, J. (ed.) (2015) 'CS4125: Serenity Gaming – a sample project from Sem1 2014-2015',
 * CS4125: Systems Analysis & Design [online],
 * available: http://www.csis.ul.ie/coursemodule/CS4125 [accessed 20 Oct 2015]
 *
 *   Adapted file: Serenity_Gaming/PanelManager.java
 *   Approximate use: 80%
 **/
package userinterface;

public class MenuManager implements UISubject {

    private MenuFactory menuFac;
    private Screen screen;
    private Menu updatedMenu;

    public MenuManager(MenuFactory menuFac) {
        this.menuFac = menuFac;
    }
```

```java
    @Override
    public void registerObserver(Screen scr) {
      screen = scr;
      getMenuFromFactory(1);
      notifyObserver();
    }

    @Override
    public void removeObserver() {
      screen = null;
    }

    @Override
    public void notifyObserver() {
      screen.update(this);
    }

    public void getMenuFromFactory(int menuID) {
      updatedMenu = menuFac.getMenu(menuID, this);
      notifyObserver();
    }

    @Override
    public Menu getUpdatedMenu() {
      return updatedMenu;
    }

    public void setUpdatedMenu(Menu menu) {
      updatedMenu = menu;
    }
}
```

MessageUI.java

```java
/*
* Anon., in Collins, J. (ed.) (2015) 'CS4125: Serenity Gaming – a sample project from Sem1 2014-2015',
* CS4125: Systems Analysis & Design [online],
* available: http://www.csis.ul.ie/coursemodule/CS4125 [accessed 20 Oct 2015]
*
*   Adapted file: Serenity_Gaming/PanelManager.java
*   Approximate use: 80%
**/
package userinterface;

public class MenuManager implements UISubject {

  private MenuFactory menuFac;
  private Screen screen;
  private Menu updatedMenu;

  public MenuManager(MenuFactory menuFac) {
    this.menuFac = menuFac;
  }

  @Override
  public void registerObserver(Screen scr) {
    screen = scr;
```

```java
        getMenuFromFactory(1);
        notifyObserver();
    }

    @Override
    public void removeObserver() {
        screen = null;
    }

    @Override
    public void notifyObserver() {
        screen.update(this);
    }

    public void getMenuFromFactory(int menuID) {
        updatedMenu = menuFac.getMenu(menuID, this);
        notifyObserver();
    }

    @Override
    public Menu getUpdatedMenu() {
        return updatedMenu;
    }

    public void setUpdatedMenu(Menu menu) {
        updatedMenu = menu;
    }
}
```

PartyUI.java

```java
/*
* Anon., in Collins, J. (ed.) (2015) 'CS4125: Serenity Gaming – a sample project from Sem1 2014-2015',
* CS4125: Systems Analysis & Design [online],
* available: http://www.csis.ul.ie/coursemodule/CS4125 [accessed 20 Oct 2015]
*
*   Adapted file: Serenity_Gaming/MenuUI.java
*   Approximate use: 30%
**/
package userinterface;

import java.awt.BorderLayout;
import java.awt.Color;
import java.awt.Dimension;
import java.awt.FlowLayout;
import java.awt.GridLayout;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.util.ArrayList;
import javax.swing.JButton;
import javax.swing.JLabel;
import javax.swing.JOptionPane;
import javax.swing.JPanel;
import javax.swing.JScrollPane;
import javax.swing.JTextArea;

public class PartyUI extends Menu {
```

```java
public PartyUI() {
    showPartyMenu();
}

public void showPartyMenu() {
    JPanel mainMenuPanel = new JPanel();
    BorderLayout mainMenuLayout = new BorderLayout();
    mainMenuPanel.setLayout(mainMenuLayout);

    JPanel topBarPanel = new JPanel();
    FlowLayout topBarLayout = new FlowLayout();
    topBarLayout.setAlignment(FlowLayout.RIGHT);
    topBarPanel.setLayout(topBarLayout);
    JLabel uiLabel = new JLabel("Party Menu");
    topBarPanel.add(uiLabel);
    JLabel spacer = new JLabel("        ");
    topBarPanel.add(spacer);
    JButton logoutButton = new JButton("Logout");
    logoutButton.addActionListener(new ActionListener() {
        @Override
        public void actionPerformed(ActionEvent e) {

            sessionInfo.logPlayerOut();
            menuMgr.getMenuFromFactory(1);

        }
    });
    topBarPanel.add(logoutButton);
    mainMenuPanel.add(topBarPanel, mainMenuLayout.NORTH);

    JPanel centerMenuPanel = new JPanel();
    BorderLayout centerMenuLayout = new BorderLayout();
    centerMenuPanel.setLayout(centerMenuLayout);
    centerMenuPanel.setBackground(Color.blue);

    // list of members and leave party button on top row
    JPanel topCenterMenuPanel = new JPanel();
    FlowLayout topCenterMenuLayout = new FlowLayout();
    topCenterMenuLayout.setAlignment(FlowLayout.LEFT);
    topCenterMenuPanel.setLayout(topCenterMenuLayout);

    // populate list with party members
    JTextArea memberList = new JTextArea();
    JScrollPane listScroller = new JScrollPane(memberList);
    listScroller.setPreferredSize(new Dimension(300, 150));
    topCenterMenuPanel.add(listScroller);
    // populate list
    populateMembersList(memberList);

    JButton leavePartyButton = new JButton("Leave Party");
    leavePartyButton.addActionListener(new ActionListener() {
        @Override
        public void actionPerformed(ActionEvent e) {
            JOptionPane.showMessageDialog(null, "You have left the party");
            sessionInfo.leaveParty();
            menuMgr.getMenuFromFactory(2);
```

```
            }
        });
        topCenterMenuPanel.add(leavePartyButton);
        centerMenuPanel.add(topCenterMenuPanel, centerMenuLayout.NORTH);

        JPanel centerMenuButtonsPanel = new JPanel();
        GridLayout centerMenuButtonsLayout = new GridLayout(4, 1);
        centerMenuButtonsPanel.setLayout(centerMenuButtonsLayout);
        JButton refreshButton = new JButton("Refresh Members List");
        refreshButton.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                sessionInfo.getPartyDetails();
                if (sessionInfo.isPlayerInParty()) {
                    populateMembersList(memberList);
                }
                else
                {
                    JOptionPane.showMessageDialog(null, "You have been removed from the party.\nReturning to
main menu", null, JOptionPane.WARNING_MESSAGE);
                    menuMgr.getMenuFromFactory(2);
                }
            }
        });
        centerMenuButtonsPanel.add(refreshButton);
        JButton inviteButton = new JButton("Invite Friend");
        inviteButton.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {

                try {
                    int friendToInvite = Integer.parseInt(JOptionPane.showInputDialog(null, "Enter the ID of the friend
you would"
                        + "\nlike to invite to join the party"));
                    if (sessionInfo.isFriend(friendToInvite)) {
                        sessionInfo.sendInvite(friendToInvite);
                    } else {
                        JOptionPane.showMessageDialog(null, "Not a valid friend ID.", null,
JOptionPane.WARNING_MESSAGE);
                    }
                } catch (Exception ex) {
                    JOptionPane.showMessageDialog(null, "Input invalid. Please enter the ID of a friend.", null,
JOptionPane.WARNING_MESSAGE);
                }
            }
        });
        centerMenuButtonsPanel.add(inviteButton);
        JButton removeMemberButton = new JButton("Remove Member");
        removeMemberButton.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                if (sessionInfo.isPartyLeader()) {
                    if (sessionInfo.getPartySize() > 1) {
                        // select member to remove
                        boolean validID = false;

                        try {
```

```java
                    int memberToRemove = Integer.parseInt(JOptionPane.showInputDialog(null, "Enter the ID of
the member you would"
                            + "\nlike to remove from the party"));
                    if(memberToRemove == sessionInfo.getPlayerId())
                    {
                        JOptionPane.showMessageDialog(null, "To leave party, please select the 'Leave Party'
button\nor log out.", null, JOptionPane.WARNING_MESSAGE);
                    }
                    else if (sessionInfo.isMemberOfParty(memberToRemove)) {

                        sessionInfo.removePlayerFromParty(memberToRemove);
                        sessionInfo.getPartyDetails();
                        menuMgr.getMenuFromFactory(3);
                    }
                } catch (Exception ex) {
                    JOptionPane.showMessageDialog(null, "Input must be an integer number", null,
JOptionPane.WARNING_MESSAGE);
                }

            } else {
                JOptionPane.showMessageDialog(null, "You are the only member in the party.\n"
                        + "To leave the party click the 'Leave Party' button or\nlog out.", null,
JOptionPane.WARNING_MESSAGE);
            }
        } else {
            JOptionPane.showMessageDialog(null, "You must be Party Leader to remove members", null,
JOptionPane.WARNING_MESSAGE);
        }
    }
});
centerMenuButtonsPanel.add(removeMemberButton);
JButton gameButton = new JButton("Games");
gameButton.addActionListener(new ActionListener() {
    @Override
    public void actionPerformed(ActionEvent e) {
        JOptionPane.showMessageDialog(null, "Component not integrated");
    }
});
centerMenuButtonsPanel.add(gameButton);
centerMenuPanel.add(centerMenuButtonsPanel, centerMenuLayout.CENTER);

mainMenuPanel.add(centerMenuPanel, mainMenuLayout.CENTER);

JPanel bottomBarPanel = new JPanel();
FlowLayout bottomBarLayout = new FlowLayout();
bottomBarLayout.setAlignment(FlowLayout.LEFT);
bottomBarPanel.setLayout(bottomBarLayout);
JButton returnButton = new JButton("<-Return");
returnButton.addActionListener(new ActionListener() {
    @Override
    public void actionPerformed(ActionEvent e) {
        menuMgr.getMenuFromFactory(2);

    }
});
bottomBarPanel.add(returnButton);
mainMenuPanel.add(bottomBarPanel, mainMenuLayout.SOUTH);
```

```java
        panel = mainMenuPanel;

    }

    public void populateMembersList(JTextArea memberList) {
        memberList.setText("");
        memberList.append("Members:\n");
        if (sessionInfo != null) {
            ArrayList<Integer> members = sessionInfo.getPartyMembers();

            memberList.append("" + members.get(0) + " (host)\n");
            for (int i = 1; i < members.size(); i++) {
                memberList.append("" + members.get(i) + "\n");
            }
        }
    }
    @Override
    public JPanel getMenuPanel() {
        return panel;
    }

    @Override
    public void setMenuManager(MenuManager menuMgr) {
        this.menuMgr = menuMgr;
    }
}
```

Screen.java

```java
/*
* Anon., in Collins, J. (ed.) (2015) 'CS4125: Serenity Gaming – a sample project from Sem1 2014-2015',
* CS4125: Systems Analysis & Design [online],
* available: http://www.csis.ul.ie/coursemodule/CS4125 [accessed 20 Oct 2015]
*
*   Adapted file: Serenity_Gaming/Window.java
*   Approximate use: 50%
**/
package userinterface;

import javax.swing.JFrame;
import javax.swing.JPanel;

public class Screen extends JFrame implements UIObserver {

    private JPanel displayedMenuPanel = null;

    public Screen() {
        super("Party maker prototype");
        setSize(500, 500);
        setResizable(false);
        setLocationRelativeTo(null);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    }

    @Override
    public void update(UISubject s) {
        dropMenuPanel();
```

```java
      Menu updatedMenu = s.getUpdatedMenu();
      displayedMenuPanel = updatedMenu.getMenuPanel();
      add(displayedMenuPanel);
      setVisible(true);
    }

    public void dropMenuPanel() {
      if (displayedMenuPanel != null) {
        remove(displayedMenuPanel);
        displayedMenuPanel = null;
      }
    }
}
```

StartScreenUI.java

```java
/*
* Anon., in Collins, J. (ed.) (2015) 'CS4125: Serenity Gaming – a sample project from Sem1 2014-2015',
* CS4125: Systems Analysis & Design [online],
* available: http://www.csis.ul.ie/coursemodule/CS4125 [accessed 20 Oct 2015]
*
*   Adapted file: Serenity_Gaming/HomeUI.java
*   Approximate use: 90%
**/
package userinterface;

import java.awt.Color;
import java.awt.Dimension;
import java.awt.GridBagConstraints;
import java.awt.GridBagLayout;
import java.awt.GridLayout;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import javax.swing.JButton;
import javax.swing.JLabel;
import javax.swing.JOptionPane;
import javax.swing.JPanel;
import javax.swing.JPasswordField;
import javax.swing.JTextField;
import javax.swing.border.Border;
import javax.swing.border.CompoundBorder;
import javax.swing.border.EmptyBorder;

public class StartScreenUI extends Menu {

  private int SCREEN_WIDTH = 0;
  private int SCREEN_HEIGHT = 0;

  public StartScreenUI() {

  }

  public StartScreenUI(int SCREEN_WIDTH, int SCREEN_HEIGHT) {
    this.SCREEN_WIDTH = SCREEN_WIDTH;
    this.SCREEN_HEIGHT = SCREEN_HEIGHT;
    showLoginScreen();
  }
```

```java
public void showLoginScreen() {
    final JPanel loginScreen = new JPanel();

    GridBagLayout windowLayout = new GridBagLayout();
    windowLayout.columnWidths = new int[]{SCREEN_WIDTH / 3, SCREEN_WIDTH / 3, SCREEN_WIDTH / 3};
    windowLayout.rowHeights = new int[]{SCREEN_HEIGHT / 3, SCREEN_HEIGHT / 3, SCREEN_HEIGHT / 3};
    windowLayout.columnWeights = new double[]{0.0, 0.0, 1.0};
    windowLayout.rowWeights = new double[]{0.0, 0.0, 1.0};
    loginScreen.setLayout(windowLayout);

    GridBagConstraints cons = new GridBagConstraints();
    cons.gridx = 1;
    cons.gridy = 1;

    JPanel loginPanel = new JPanel();

    Border margin = new EmptyBorder(10, 10, 10, 10);
    Border loginBorder = loginPanel.getBorder();
    loginPanel.setBorder(new CompoundBorder(loginBorder, margin));

    loginPanel.setBackground(Color.gray);
    GridLayout loginLayout = new GridLayout(3, 2);
    loginLayout.setHgap(10);
    loginLayout.setVgap(10);
    loginPanel.setLayout(loginLayout);
    JLabel usernameLabel = new JLabel("Username:");
    loginPanel.add(usernameLabel);
    final JTextField getUsername = new JTextField();
    loginPanel.add(getUsername);
    JLabel passwordLabel = new JLabel("Password:");
    loginPanel.add(passwordLabel);
    final JPasswordField getPassword = new JPasswordField();
    loginPanel.add(getPassword);
    JButton loginButton = new JButton("Login");
    loginPanel.add(loginButton);
    JButton registerButton = new JButton("Register");
    loginPanel.add(registerButton);
    loginScreen.add(loginPanel, cons);

    loginButton.addActionListener(new ActionListener() {
        @Override
        public void actionPerformed(ActionEvent e) {

            String username = getUsername.getText().toLowerCase();
            String password = getPassword.getText();
            if (!username.equals("") && !password.equals("")) {
                boolean login = sessionInfo.canUserLogin(username, password);

                if (login) {
                    menuMgr.getMenuFromFactory(2);
                    System.out.println("logged in");
                } else {
                    JOptionPane.showMessageDialog(null, "Invalid login details");
                    getUsername.setText("");
                    getPassword.setText("");
                }
```

```
            } else {
              JOptionPane.showMessageDialog(null, "Fields Blank");
            }
          }
      });

      registerButton.addActionListener(new ActionListener() {
        @Override
        public void actionPerformed(ActionEvent e) {

          loginScreen.setVisible(false);
          showRegisterScreen();
          menuMgr.notifyObserver();
        }
      });

      panel = loginScreen;
  }

  public void showRegisterScreen() {
      final JPanel registerPanel = new JPanel();

      GridBagLayout windowLayout = new GridBagLayout();
      windowLayout.columnWidths = new int[]{SCREEN_WIDTH / 4, SCREEN_WIDTH / 2, SCREEN_WIDTH / 4};
      windowLayout.rowHeights = new int[]{SCREEN_HEIGHT / 4, SCREEN_HEIGHT / 2, SCREEN_HEIGHT / 4};
      windowLayout.columnWeights = new double[]{0.0, 0.0, 1.0};
      windowLayout.rowWeights = new double[]{0.0, 0.0, 1.0};
      registerPanel.setLayout(windowLayout);

      GridBagConstraints cons = new GridBagConstraints();
      cons.weighty = 1.0;
      cons.weightx = 1.0;
      cons.gridx = 1;
      cons.gridy = 1;
      cons.gridheight = 1;
      cons.gridwidth = 1;

      JPanel registerUserPanel = new JPanel();
      registerUserPanel.setMinimumSize(new Dimension(SCREEN_WIDTH / 2 + 20, SCREEN_HEIGHT / 2));
      Border margin = new EmptyBorder(10, 10, 10, 10);
      Border loginBorder = registerUserPanel.getBorder();
      registerUserPanel.setBorder(new CompoundBorder(loginBorder, margin));
      registerUserPanel.setBackground(Color.lightGray);

      GridLayout registerUserLayout = new GridLayout(7, 2);
      registerUserLayout.setHgap(10);
      registerUserLayout.setVgap(10);
      registerUserPanel.setLayout(registerUserLayout);

      final JLabel errorLabel = new JLabel("");
      errorLabel.setForeground(Color.red);
      registerUserPanel.add(errorLabel);
      registerUserPanel.add(new JLabel(""));
      JLabel usernameLabel = new JLabel("Username:");
      registerUserPanel.add(usernameLabel);
      final JTextField getUsername = new JTextField();
      registerUserPanel.add(getUsername);
```

```java
JLabel emailLabel = new JLabel("Email:");
registerUserPanel.add(emailLabel);
final JTextField getEmail = new JTextField();
registerUserPanel.add(getEmail);
JLabel passwordLabel = new JLabel("Password:");
registerUserPanel.add(passwordLabel);
final JPasswordField getPassword = new JPasswordField();
registerUserPanel.add(getPassword);
JLabel confirmPasswordLabel = new JLabel("Confirm Password:");
registerUserPanel.add(confirmPasswordLabel);
final JPasswordField getConfirmPassword = new JPasswordField();
registerUserPanel.add(getConfirmPassword);
JButton createButton = new JButton("Create");
createButton.addActionListener(new ActionListener() {
    @Override
    public void actionPerformed(ActionEvent e) {
        /* Regex to ensure email given is correct format */
        String emailPattern = "^[_A-Za-z0-9-\\+]+(\\.[_A-Za-z0-9-]+)*@"
            + "[A-Za-z0-9-]+(\\.[A-Za-z0-9]+)*(\\.[A-Za-z]{2,})$";
        String passwordPattern = "";
        String username = getUsername.getText().toLowerCase();
        String email = getEmail.getText().toLowerCase();
        String password = getPassword.getText();
        String confirmPassword = getConfirmPassword.getText();

        if (!username.equals("") && !email.equals("")
                && !password.equals("") && !confirmPassword.equals("")) {
            if (password.equals(confirmPassword)) {
                if (email.matches(emailPattern)) {
                    /* Checks if username and email are free to use */
                    int checkUsernameEmail = sessionInfo.checkUsernameEmail(username, email);
                    /* Username and email are free */
                    if (checkUsernameEmail == 2) {
                        sessionInfo.createPlayer(username, password, email);
                        JOptionPane.showMessageDialog(null, "User created!");
                        registerPanel.setVisible(false);
                        showLoginScreen();
                        menuMgr.notifyObserver();
                    } else {
                        /* Username already registered */
                        if (checkUsernameEmail == 0) {
                            errorLabel.setText("Username already in use");
                        } /* Email already registered */ else {
                            errorLabel.setText("Email already in use");
                        }
                    }
                } /* Email does not match pattern */ else {
                    errorLabel.setText("Not a valid email address");
                }
            } /* Passwords do not match */ else {
                errorLabel.setText("Incorrect Passwords");
            }
        } /* Fields left blank */ else {
            errorLabel.setText("Field(s) blank");
        }
    }
});
```

```
      registerUserPanel.add(createButton);
      /* User cancels registration, redirected back to login screen */
      JButton cancelButton = new JButton("Cancel");
      cancelButton.addActionListener(new ActionListener() {
        @Override
        public void actionPerformed(ActionEvent e) {
          registerPanel.setVisible(false);
          showLoginScreen();
          menuMgr.notifyObserver();
        }
      });
      registerUserPanel.add(cancelButton);

      registerPanel.add(registerUserPanel, cons);
      panel = registerPanel;
    }

    @Override
    public JPanel getMenuPanel() {
      return this.panel;
    }
}
```

StartUpUI.java

```
/*
* Anon., in Collins, J. (ed.) (2015) 'CS4125: Serenity Gaming – a sample project from Sem1 2014-2015',
* CS4125: Systems Analysis & Design [online],
* available: http://www.csis.ul.ie/coursemodule/CS4125 [accessed 20 Oct 2015]
*
*  Adapted file: Serenity_Gaming/BootUI.java
*  Approximate use: 85%
**/
package userinterface;

import database.DatabaseAccess;
import database.DatabaseInterface;
import session.SessionInformation;

public class StartUpUI {

  public StartUpUI() {

  }

  public void run() {
    try {
      /* Creates window to display GUI components */
      /* Observer to PanelManager */
      Screen screen = new Screen();
      /* Starts database connection */
      DatabaseInterface database = new DatabaseAccess();
      /* Will process input taken from client GUI */
      SessionInformation sessionInfo = SessionInformation.getInstance();
      sessionInfo.setDbConnection(database);
      /* Panel factory to display panels on window */
      MenuFactory menuFac = new MenuFactory(sessionInfo);
```

```
        /* Subject in observer pattern */
        MenuManager menuMgr = new MenuManager(menuFac);
        menuMgr.registerObserver(screen);
      } catch (Exception e) {
        System.out.println("Error bootingUI: " + e.toString());
      }
    }
  }
}
```

UIObserver.java

```
/*
* Anon., in Collins, J. (ed.) (2015) 'CS4125: Serenity Gaming – a sample project from Sem1 2014-2015',
* CS4125: Systems Analysis & Design [online],
* available: http://www.csis.ul.ie/coursemodule/CS4125 [accessed 20 Oct 2015]
*
*   Adapted file: Serenity_Gaming/Observer.java
*   Approximate use: 20%
**/
package userinterface;

public interface UIObserver
{
   public abstract void update(UISubject s);
}
```

UISubject.java

```
/*
* Anon., in Collins, J. (ed.) (2015) 'CS4125: Serenity Gaming – a sample project from Sem1 2014-2015',
* CS4125: Systems Analysis & Design [online],
* available: http://www.csis.ul.ie/coursemodule/CS4125 [accessed 20 Oct 2015]
*
*   Adapted file: Serenity_Gaming/Subject.java
*   Approximate use: 70%
**/
package userinterface;

public interface UISubject {

   public abstract void registerObserver(Screen scr);

   public abstract void removeObserver();

   public abstract void notifyObserver();

   public abstract Menu getUpdatedMenu();
}
```

Driver Package

AppDriver.java

```
package driver;

import userinterface.StartUpUI;


public class AppDriver {
```

```java
    public static void main(String[] args)
    {
        /* Main: Boots User Interface */
        StartUpUI startUp = new StartUpUI();
        startUp.run();
    }
}
```

# Design Artefacts

Architectural Diagram:
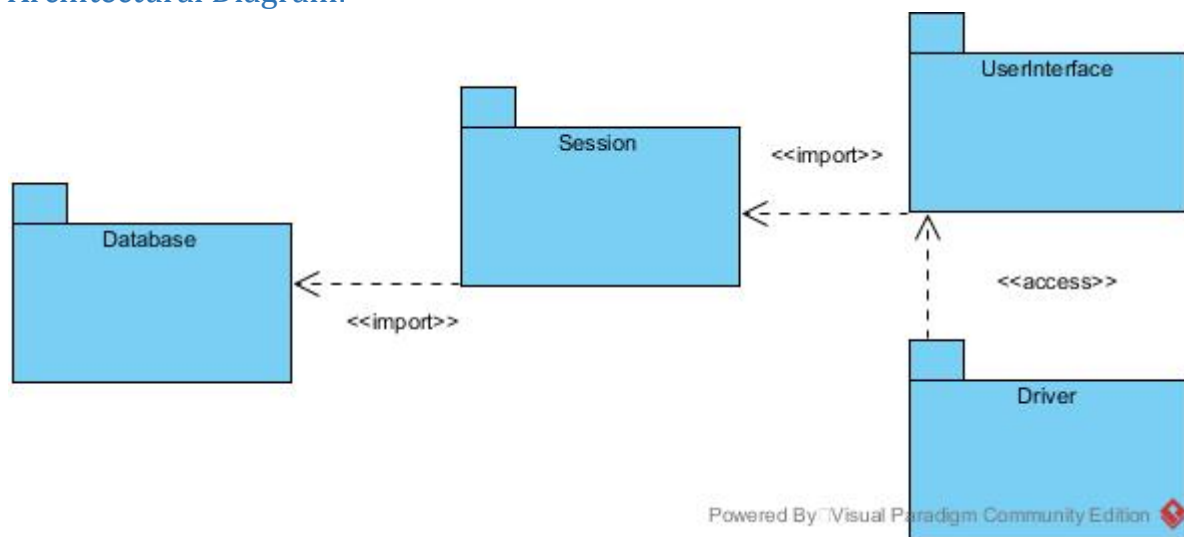
Class Diagram:
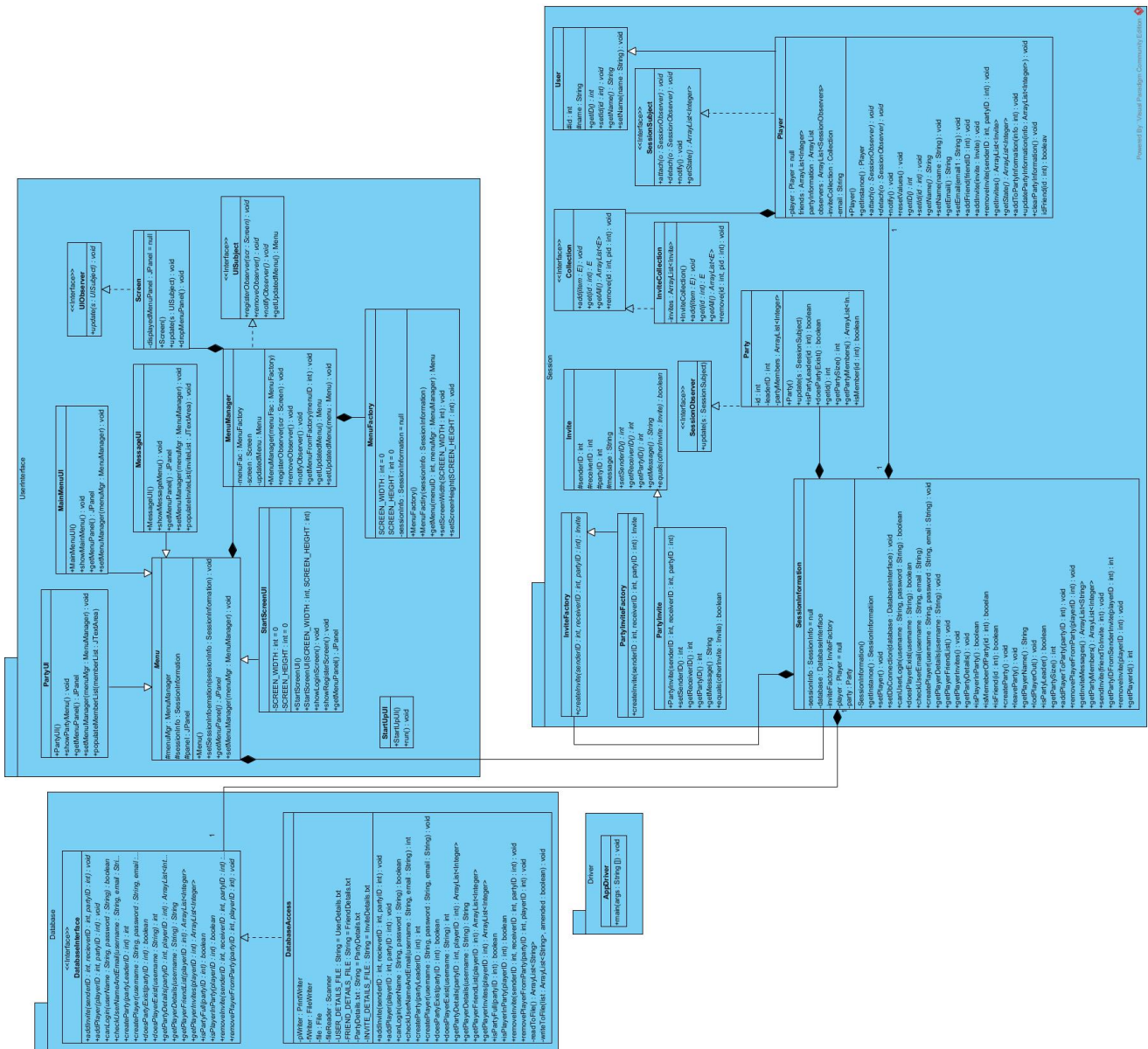
## Sequence Diagram:
Refresh Invite List:
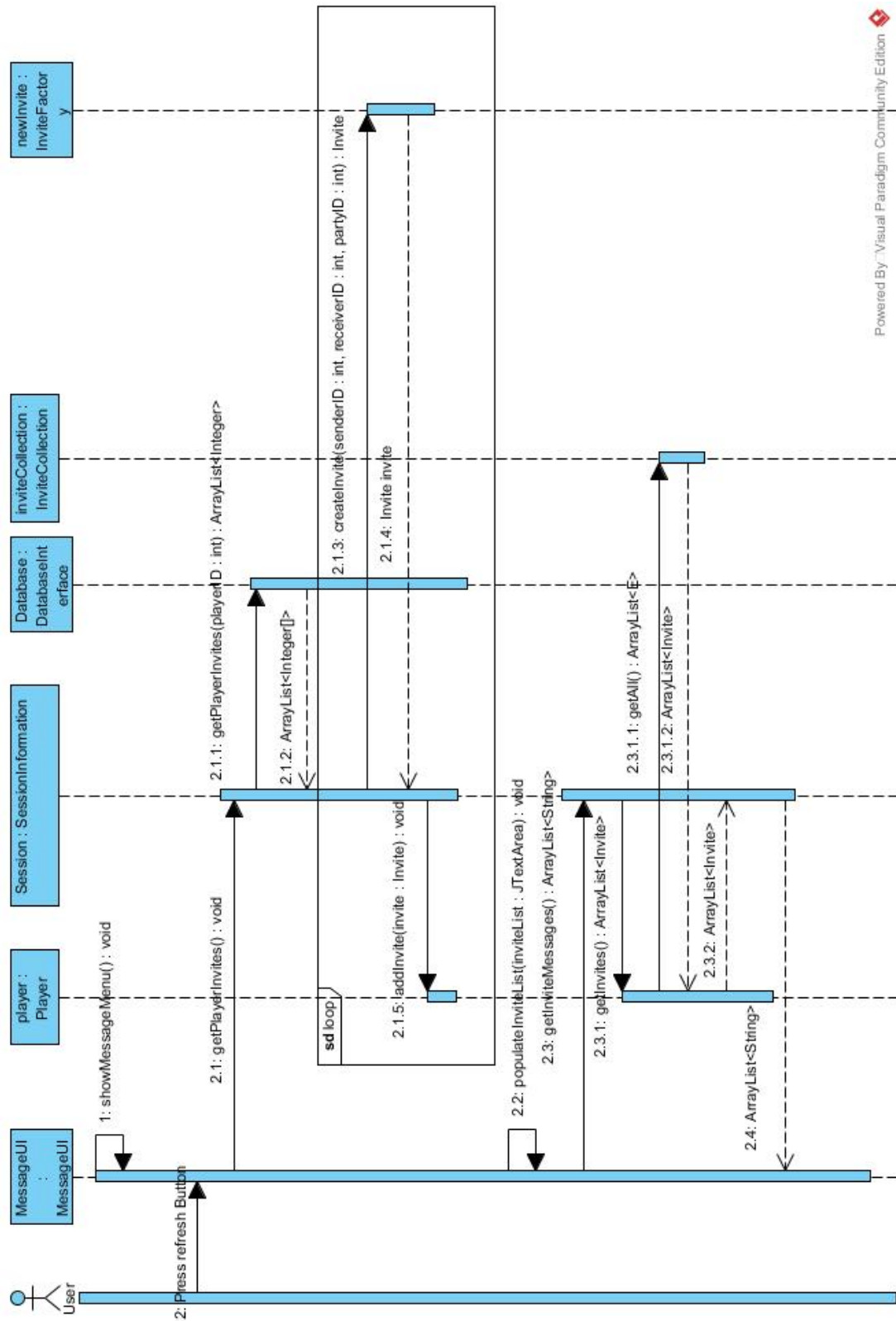
State Diagrams:

Database Manager:
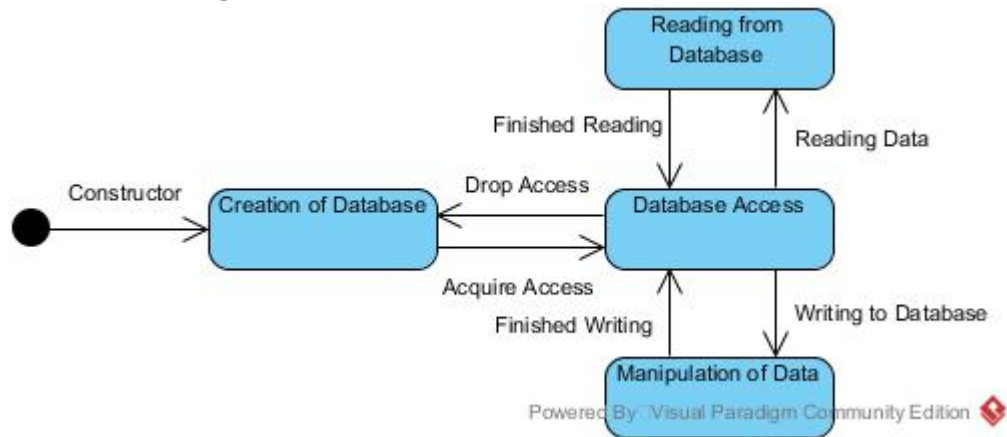
Observer State:
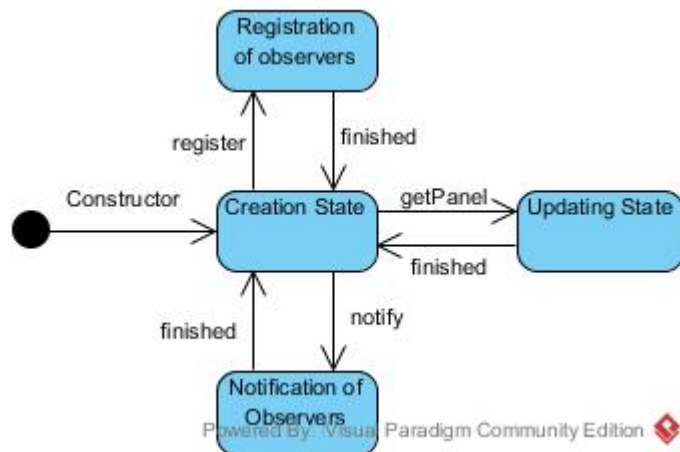
Design Patterns

We have implemented a number of design patterns into this system. The application's user interface implements both the observer design pattern and the factory design pattern. The implementation used to create invites uses the factory design method. The observer pattern is also used to express the relationship between the instance of Player.java and Party.java. The factory design pattern belongs to the creational pattern family. This design pattern allows for the abstraction of implementation logic and allows this to be delegated to concrete subclasses. This provides an excellent method for extension of a software. We applied the factory design pattern in the user interface as a means to create all the necessary screens that were needed to be displayed to the user. We also implemented this design pattern for handling the creation of invites that a user can send to other users. For this prototype, we only implemented this with the party invite feature, but in future iterations of the application, the use of the factory design will allow the addition of extra invite functionality such as: group invites, friend invites, and messaging, etc.

The observer pattern, which is a behavioural design pattern defines a one to many relationship between a subject object and one or more observer objects. When the subject changes state it notifies the observers that are dependent on it. With regards to the user interface, this allows us to display any new menu instantiated by the menu factory automatically to the screen as the screen is an observer of the menu factory. The interfaces UIObserver.java and UISubject.java are used to implement this design in the user interface. It is realised through MenuManager.java and Screen.java. In the Session package, we also implemented the observer pattern to reflect changes to the player's party status. When a change is detected in the Party object it notifies it's observer (in this case an instance of Party.java) which updates it's state accordingly. This allows for party information such as the party's ID, Leader ID and members in the party to be automatically updated as it changes. The observer pattern for player to party interaction is interfaced by SessionObserver.java and SessionSubject.java.

As we only ever want one instance of Player.java per session, we have incorporated the singleton pattern to the player object. This creational pattern ensures that only one instance of a class can be instantiated. We have achieved this by making the access modifier of the constructor to the class private and using a static method to instantiate the object. We have also incorporated this design pattern into the screen class as we only required one instance to be realised. Examples of the singleton pattern can be found in Player.java, Screen.java, SessionInformation.java and Party.java.

Critique

Analysis Artefacts and Design – Compare and Contrast:

When we compared the artefacts from the analysis stage with the end result, it was found that while a lot of the key features remained recognisable, overall the changes that had taken place and the refinements brought about, made the system a better example of where we would take the project if we were to take it further. Currently there is quite a high amount of cohesion (coupling) in the system. This is due to us being slightly blindsided when attempting to make the User Interface a more defined and visible thing, rather than worrying about how it looked to the user. Our attentions should have been focused more clearly on what the system did.

We feel that while we did suffer a bit from paralysis by analysis at first, once we approached the system in a more Agile manner, our focus was concentrated and overcame the paralysis.

Server and Network Communications:

For further development, we would like to incorporate a Server and the communications needed to interact with the client applications. This would enable us to control vital aspects of the system, for example: live feeds of what information is made available to the user, what is displayed as Adverts, if adverts were decided upon in the future. At present the application is a standalone application, which interacts briefly with a "simulation" database. We would look to incorporate our traffic around pre existing protocols like HTTP, as this would reduce the cost in implementation and enable us to make use of the server functionality with a Restful API. Implementing the system in this way eliminates the need to deal with sockets and significantly reduces coupling since proprietary objects would not be needed, this approach would also aid language independence and make client implementations easier.

Database - Server Communication:

One of the aspects of the system that we would have liked to get a more finished example completed was the database. Due to time constraints and work-load on the team, we were unable to successfully get a SQL database setup and interacting with our system. At present we simulate the actions that would take place in a class called DatabaseAccess.java using .txt files, with read and write operations performed on them, instead of insertions etc. on a SQL table or database.

User-Interface:

The user interface was one of the areas we found ourselves approaching in a more code and fix way. We were distracted from providing core functionality in attempting to ensure we had something that could visually be shown with interactions with the end user. The user Interface package is quite a robust package, which tends to suffer from coupling issues. While the user interface is easy to navigate due to the unforeseen time sink that took place on this element, our overall functionality suffered.

Session:

The session package of the system manages the "under-the-hood" functionality of the system implemented and monitors for changes in states from elsewhere. The Session-Observer waits for any signals emitted from any of the package classes and updates others accordingly. The Session–Subject attaches the observer and detaches as needed. The User class is an abstract parent class of every User of the system while the player is the more specific child of the User class. The player class also implements the Session-Subject Interface.

The Invite subsystem is an implementation of some Interface classes and a defined Collection. We aimed to design out system to interfaces rather than implementation as best we could, and we believe our implementation of the Session package is a good example of this methodology.

Concurrency:

Although we failed to tackle the issue of concurrency in the final implementation of the code, we did decide on some ways we would tackle the issue in further iterations of the system. We would look to apply thread and thread management to: the invite and messages subsystem, the player class, and the party subsystem. The threading on the invite and messages subsystem would check for and changes in messages and state of message inbox, listen for any messages being sent from other users. The player class would be threaded to enable a faster response in passing updates to the observer and monitoring for any up to date state changes. The threading of the party subsystem would manage the current user's party and any further invites to join new parties, merge existing parties and update on party members currently in the existing party. We would look to handle any issues with multiple requests through programming to prevention invoking the wound - wait algorithm.

References

Anon., in Collins, J. (ed.) (2015) 'CS4125: Serenity Gaming – a sample project from Sem1 2014-2015', CS4125: Systems Analysis & Design [online], available: http://www.csis.ul.ie/coursemodule/CS4125 [accessed 20 Oct 2015]

Aplica (2015) methodology [image online], available: http://www.aplicatech.com/agile-project-management-aplica [accessed 25 Nov 2015]

Business Analyst Mentor (2015) vmodel.jpg [image online], available: http://businessanalystmentor.com/user-acceptance-testing-business-analyst-perspective/ [accessed 25 Nov 2015]

Mohammed Munassar Govardhan, N, A, A (2010), "A Comparison Between Five Models Of Software Engineering ", IJCSI International Journal of Computer Science Issues, Vol. 7, Issue 5, September 2010 ISSN[Online], Available: 1694-0814 http://www.IJCSI.org [accessed 17 Nov 2015]

Parnas, David L.; Clements, Paul C. (1986). "A rational design process: How and why to fake it" (PDF). Software Engineering, IEEE Transactions: 251–257, Available: http://www.cs.tufts.edu/~nr/cs257/archive/david-parnas/fake-it.pdf [accessed 21 Nov 2015]

RailsHorde (2015) waterfall_copy.jpg [image online], available: http://www.railshorde.com/blog/waterfall-development-model [accessed 25 Nov 2015]

Appendix A

DatabaseInterface.java

```java
/*
* Anon., in Collins, J. (ed.) (2015) 'CS4125: Serenity Gaming – a sample project from Sem1 2014-2015',
* CS4125: Systems Analysis & Design [online],
* available: http://www.csis.ul.ie/coursemodule/CS4125 [accessed 20 Oct 2015]
*/
import java.util.ArrayList;

public interface DatabaseInterface {


        boolean canLogin(String username, String password) throws Exception;

        /* References the local Player class */
        String getPlayerDetails(String username) throws Exception;

        ArrayList<String> getPlayerFriendList(int playerID) throws Exception;

        ArrayList<String> getPlayerMessages(int playerID) throws Exception;

        ArrayList<String> getPlayerInvites(int playerID) throws Exception;

        void createPlayer(String username,String password, String email) throws Exception;

        int createParty(int partyLeaderID) throws Exception;

        String [] getPartyDetails(int partyID) throws Exception;

        boolean isPartyFull(int partyID) throws Exception;

        void addPlayerToParty(int playerID, int partyID) throws Exception;

        void removePlayerFromParty(int partyID, int playerID) throws Exception;

        int checkUserNameAndEmail(String username, String email) throws Exception;

        boolean doesPartyExist(int partyID) throws Exception;

        int doesPlayerExist(String username) throws Exception;

        boolean isPlayerInParty(int playerID) throws Exception;

        void addNewInvite(int senderID, String senderName, int receiverID, int inviteType) throws Exception;

        void addNewInvite(int senderID, String senderName, int receiverID, int inviteType, int partyID) throws Exception;

        void addFriends(int senderID, int receiverID) throws Exception;

        void createMessage(int senderID, String senderName, int receiverID, String message) throws Exception;

        void removeMessage(int messageID) throws Exception;
}
```

DatabaseAccess.java

```java
/*
* Anon., in Collins, J. (ed.) (2015) 'CS4125: Serenity Gaming – a sample project from Sem1 2014-2015',
* CS4125: Systems Analysis & Design [online],
* available: http://www.csis.ul.ie/coursemodule/CS4125 [accessed 20 Oct 2015]
*/
import java.io.PrintWriter;
import java.io.FileWriter;
import java.io.File;
import java.util.Scanner;
import java.util.ArrayList;
```

```java
import java.util.Arrays;
import java.util.List;


public class DatabaseAccess implements DatabaseInterface {

        private PrintWriter pWriter;
        private FileWriter fWriter;
        private File file;
        private Scanner fileReader;
        private final String USER_DETAILS_FILE = "UserDetails.txt";
        private final String FRIEND_DETAILS_FILE = "FriendDetails.txt";
        private final String MESSAGE_DETAILS_FILE = "MessageDetails.txt";
        private final String PARTY_DETAILS_FILE = "PartyDetails.txt";
        private final String INVITE_DETAILS_FILE = "InviteDetails.txt";

        public DatabaseAccess() throws Exception
        {

        }

        @Override
        public boolean canLogin(String username, String password) throws Exception {
                file = new File(USER_DETAILS_FILE);
                if(file.exists())
                {
                        fileReader = new Scanner(file);
                        while(fileReader.hasNextLine())
                        {
                                String [] lineFromFile = (fileReader.nextLine()).split(",");
                                if(lineFromFile[2].equals(password))
                                {
                                        fileReader.close();
                                        file = null;
                                        return true;
                                }
                        }
                }

                fileReader.close();
                file = null;
                return false;
        }

        @Override
        public String getPlayerDetails(String  username) throws Exception {
                file = new File(USER_DETAILS_FILE);
                fileReader = new Scanner(file);
                String playerDetails = "";
                while(fileReader.hasNextLine())
                {
                        String [] lineFromFile = (fileReader.nextLine()).split(",");
                        if(lineFromFile[1].equals(username))
                        {
                                playerDetails += lineFromFile[0] + ",";
                                playerDetails += lineFromFile[1] + ",";
                                playerDetails += lineFromFile[3] + ",";
                        }
                }
                fileReader.close();
                file = null;
                return playerDetails;
        }

        @Override
        public ArrayList<String> getPlayerFriendList(int playerID) throws Exception
        {
                String pidStr = playerID + "";
                ArrayList<String> friendIDs = new ArrayList<String>();
                ArrayList<String> friendDetails = new ArrayList<String>();
                file = new File(FRIEND_DETAILS_FILE);
```

```java
                fileReader = new Scanner(file);
                while(fileReader.hasNextLine())
                {
                        String lineFromFile [] = (fileReader.nextLine()).split(",");
                        if(pidStr.equals(lineFromFile[0]) || pidStr.equals(lineFromFile[1]))
                        {
                                if(pidStr.equals(lineFromFile[0]))
                                {
                                        friendIDs.add(lineFromFile[1]);
                                }
                                else
                                {
                                        friendIDs.add(lineFromFile[0]);
                                }
                        }
                }
                fileReader.close();
                file = new File(USER_DETAILS_FILE);
                fileReader = new Scanner(file);
                while(fileReader.hasNextLine())
                {
                        String lineFromFile [] = (fileReader.nextLine()).split(",");
                        if(friendIDs.contains(lineFromFile[0]))
                        {
                                friendDetails.add(lineFromFile[0] + "," + lineFromFile[1]);
                        }
                }
                fileReader.close();
                file = null;

                return friendDetails;
        }

        @Override
        public ArrayList<String> getPlayerMessages(int playerID) throws Exception
        {
                String pidStr = playerID + "";
                file = new File(MESSAGE_DETAILS_FILE);
                fileReader = new Scanner(file);
                ArrayList<String> messages = new ArrayList<String>();
                while(fileReader.hasNextLine())
                {
                        String [] lineFromFile = (fileReader.nextLine()).split(",", 5);
                        if(pidStr.equals(lineFromFile[1]))
                        {
                                messages.add(lineFromFile[0] + "," + lineFromFile[1] + "," + lineFromFile[2] + "," +
lineFromFile[3] + "," + lineFromFile[4]);
                        }
                }
                fileReader.close();
                file = null;
                return messages;
        }

        @Override
        public ArrayList<String> getPlayerInvites(int playerID) throws Exception
        {
                String pidStr = playerID + "";
                file = new File(INVITE_DETAILS_FILE);
                fileReader = new Scanner(file);
                ArrayList<String> invites = new ArrayList<String>();
                while(fileReader.hasNextLine())
                {
                        String [] lineFromFile = (fileReader.nextLine()).split(",");
                        if(pidStr.equals(lineFromFile[3]))
                        {
                                invites.add(lineFromFile[0] + "," + lineFromFile[1] + "," + lineFromFile[2] + "," + lineFromFile[3]
+ "," + lineFromFile[4]);
                        }
                }
                fileReader.close();
```

```java
            file = null;
            return invites;
    }

    @Override
    public void createPlayer(String username,String password,String email) throws Exception {
            file = new File(USER_DETAILS_FILE);
            fileReader = new Scanner(file);
            ArrayList<String> userDetails = new ArrayList<String>();
            while(fileReader.hasNextLine())
            {
                    userDetails.add(fileReader.nextLine());
            }
            fileReader.close();
            String [] lastEntry = (userDetails.get(userDetails.size() - 1)).split(",");
            int newPlayerID = Integer.parseInt(lastEntry[0]) + 1;
            String playerDetails = newPlayerID + "," + username + "," + password + "," + email;

            fWriter = new FileWriter(file, true);
            pWriter = new PrintWriter(fWriter);
            pWriter.println(playerDetails);
            pWriter.close();
            fWriter.close();
            file = null;
    }

    @Override
    public int createParty(int partyLeaderID) throws Exception {
            String newParty = "";
            ArrayList<String> parties = new ArrayList<String>();
            file = new File(PARTY_DETAILS_FILE);
            fileReader = new Scanner(file);
            while(fileReader.hasNextLine())
            {
                    parties.add(fileReader.nextLine());
            }
            fileReader.close();
            int partyID = parties.size() + 1;
            newParty += partyID + "," + partyLeaderID;

            fWriter = new FileWriter(file, true);
            pWriter = new PrintWriter(fWriter);
            pWriter.println(newParty);
            pWriter.close();
            fWriter.close();
            file = null;

            return partyID;
    }
    @Override
    public String [] getPartyDetails(int partyID) throws Exception
    {

            file = new File(PARTY_DETAILS_FILE);
            fileReader = new Scanner(file);
            while(fileReader.hasNextLine())
            {
                    String lineFromFile [] = fileReader.nextLine().split(",");
                    if(partyID == Integer.parseInt(lineFromFile[0]))
                    {
                            fileReader.close();
                            file = null;
                            return lineFromFile;
                    }
            }
            return null;
    }
    @Override
    public boolean isPartyFull(int partyID) throws Exception
    {
            boolean partyFull = true;
```

```java
            file = new File(PARTY_DETAILS_FILE);
            fileReader = new Scanner(file);
            while(fileReader.hasNextLine())
            {
                    String lineFromFile [] = fileReader.nextLine().split(",");
                    if(partyID == Integer.parseInt(lineFromFile[0]))
                    {
                            if(lineFromFile.length < 6)
                            {
                                    partyFull = false;
                                    break;
                            }
                    }
            }
            fileReader.close();
            file = null;
            return partyFull;
    }


    @Override
    public void addPlayerToParty(int playerID, int partyID) throws Exception
    {
            ArrayList<String> parties = new ArrayList<String>();
            file = new File(PARTY_DETAILS_FILE);
            fileReader = new Scanner(file);
            while(fileReader.hasNextLine())
            {
                    parties.add(fileReader.nextLine());
            }
            fileReader.close();
            for(int i = 0; i < parties.size(); i++)
            {
                    String [] lineFromFile = parties.get(i).split(",");
                    if(partyID == Integer.parseInt(lineFromFile[0]))
                    {
                            String amendedParty = parties.get(i);
                            amendedParty += "," + playerID;
                            parties.remove(i);
                            parties.add(i, amendedParty);
                            break;
                    }
            }
            fWriter = new FileWriter(file);
            pWriter = new PrintWriter(fWriter);
            for(int i = 0; i < parties.size(); i++)
            {
                    pWriter.println(parties.get(i));
            }
            pWriter.close();
            fWriter.close();
            file = null;
    }


    @Override
    public void removePlayerFromParty(int partyID, int playerID) throws Exception {
            ArrayList<String> parties = new ArrayList<String>();
            file = new File(PARTY_DETAILS_FILE);
            fileReader = new Scanner(file);
            while(fileReader.hasNextLine())
            {
                    parties.add(fileReader.nextLine());
            }
            fileReader.close();
            for(int i = 0; i < parties.size(); i++)
            {
                    String amendedParty = "";
                    String [] partyInfo = (parties.get(i)).split(",");
                    if(partyID == Integer.parseInt(partyInfo[0]))
                    {
                            amendedParty += partyInfo[0];
                            for(int j = 1; j < partyInfo.length;j++)
```

```java
                        {
                                if(playerID != Integer.parseInt(partyInfo[j]))
                                {
                                        amendedParty += "," + partyInfo[j];
                                }
                        }
                }
                parties.remove(i);
                if(amendedParty.contains(","))
                {
                        parties.add(i, amendedParty);
                }
                break;
        }
        file.delete();
        fWriter = new FileWriter(file, false);
        pWriter = new PrintWriter(fWriter);
        for(int i = 0; i < parties.size(); i++)
        {
                pWriter.println(parties.get(i));
        }
        pWriter.close();
        fWriter.close();
        file = null;
}


@Override
public int checkUserNameAndEmail(String username, String email) throws Exception {
        file = new File(USER_DETAILS_FILE);
        fileReader = new Scanner(file);

        while(fileReader.hasNextLine())
        {
                String [] lineFromFile = (fileReader.nextLine()).split(",");
                if(lineFromFile[1].equals(username))
                {
                        fileReader.close();
                        file = null;
                        return 0;
                }
                else if(lineFromFile[3].equals(email))
                {
                        fileReader.close();
                        file = null;
                        return 1;
                }
        }
        fileReader.close();
        file = null;
        return 2;
}

@Override
public boolean doesPartyExist(int partyID) throws Exception
{
        file = new File(PARTY_DETAILS_FILE);
        fileReader = new Scanner(file);
        boolean partyExists = false;
        while(fileReader.hasNextLine())
        {
                String [] lineFromFile = fileReader.nextLine().split(",");
                if(partyID == Integer.parseInt(lineFromFile[0]));
                {
                        partyExists = true;
                }
        }
        fileReader.close();
        file = null;
        return partyExists;
}
```

```java
@Override
public int doesPlayerExist(String username) throws Exception
{
        file = new File(USER_DETAILS_FILE);
        fileReader = new Scanner(file);
        while(fileReader.hasNextLine())
        {
                String [] lineFromFile = fileReader.nextLine().split(",");
                if(username.equals(lineFromFile[1]))
                {
                        fileReader.close();
                        file = null;
                        return Integer.parseInt(lineFromFile[0]);
                }
        }
        fileReader.close();
        file = null;
        return 0;
}

@Override
public boolean isPlayerInParty(int playerID) throws Exception
{
        boolean isInParty = false;
        file = new File(PARTY_DETAILS_FILE);
        fileReader = new Scanner(file);
        while(fileReader.hasNextLine())
        {
                String [] lineFromFile = fileReader.nextLine().split(",");
                for(int i = 1; i < lineFromFile.length; i++)
                {
                        if(playerID == Integer.parseInt(lineFromFile[i]))
                        {
                                isInParty = true;
                                break;
                        }
                }
        }
        fileReader.close();
        file = null;

        return isInParty;
}

public void addNewInvite(int senderID, String senderName, int receiverID, int inviteType) throws Exception
{
        addNewInvite(senderID, senderName, receiverID, inviteType, 0);
}

public void addNewInvite(int senderID, String senderName, int receiverID, int inviteType, int partyID) throws Exception
{
        file = new File(INVITE_DETAILS_FILE);
        fileReader = new Scanner(file);
        ArrayList<String> invites = new ArrayList<String>();
        String newInvite = "";
        while(fileReader.hasNextLine())
        {
                invites.add(fileReader.nextLine());
        }
        fileReader.close();
        int newInviteID = invites.size() + 1;
        newInvite += newInviteID + "," + senderID + "," + senderName + "," + receiverID + "," + inviteType;
        if(partyID != 0)
        {
                newInvite += "," + partyID;
        }
        fWriter = new FileWriter(file, true);
        pWriter = new PrintWriter(fWriter);
        pWriter.println(newInvite);
        pWriter.close();
```

```java
                fWriter.close();
                file = null;
        }

        public void addFriends(int senderID, int receiverID) throws Exception
        {
                String newFriendEntry = "" + senderID + "," + receiverID;
                file = new File(FRIEND_DETAILS_FILE);
                fWriter = new FileWriter(file, true);
                pWriter = new PrintWriter(fWriter);

                pWriter.println(newFriendEntry);

                pWriter.close();
                fWriter.close();
                file = null;
        }

        public void createMessage(int senderID, String senderName, int receiverID, String message) throws Exception
        {
                file = new File(MESSAGE_DETAILS_FILE);
                fileReader = new Scanner(file);
                ArrayList<String> messages = new ArrayList<String>();
                String newMessage = "";
                while(fileReader.hasNextLine())
                {
                        messages.add(fileReader.nextLine());
                }
                fileReader.close();
                int newMessageID = messages.size() + 1;
                newMessage += newMessageID + "," + senderID + "," + senderName + "," + receiverID + "," + message;
                fWriter = new FileWriter(file, true);
                pWriter = new PrintWriter(fWriter);
                pWriter.println(newMessage);
                pWriter.close();
                fWriter.close();
                file = null;
        }

        public void removeMessage(int messageID) throws Exception
        {
                file = new File(MESSAGE_DETAILS_FILE);
                fileReader = new Scanner(file);
                ArrayList<String> messages = new ArrayList<String>();
                while(fileReader.hasNextLine())
                {
                        String [] lineFromFile = fileReader.nextLine().split(",", 5);
                        if(messageID != Integer.parseInt(lineFromFile[0]))
                        {
                                String msg = lineFromFile[0] + "," + lineFromFile[1] + "," + lineFromFile[2] + "," +
lineFromFile[3]
                                        + "," + lineFromFile[4];
                                messages.add(msg);
                        }
                }
                fileReader.close();
                fWriter = new FileWriter(file);
                pWriter = new PrintWriter(fWriter);
                for(int i = 0; i < messages.size(); i++)
                {
                        pWriter.println(messages.get(i));
                }
                pWriter.close();
                fWriter.close();
                file = null;
        }
}
```

Collection.java

```java
import java.util.ArrayList;
```

```java
public interface Collection<E>
{
        public abstract void add(E item);

        public abstract E get(int id);

        public abstract ArrayList<E> getAll();

        public abstract void remove(int id);
}
```

Friend.java

```java
public class Friend extends User
{
        public Friend()
        {

        }

        public int getId()
        {
                return id;
        }

        public void setId(int id)
        {
                this.id = id;
        }

        public String getName()
        {
                return name;
        }

        public void setName(String name)
        {
                this.name = name;
        }

}
```

FriendCollection.java

```java
import java.util.ArrayList;

public class FriendCollection implements Collection <User>
{
        private ArrayList<User> friendList = null;

        public FriendCollection()
        {
                friendList = new ArrayList<User>();
        }

        public void add(User friend)
        {
                boolean newFriend = true;
                for(int i = 0; i < friendList.size(); i++)
                {
                        if(friend.getId() == (friendList.get(i)).getId())
                        {
                                newFriend = false;
                                break;
                        }
                }
                if(newFriend)
                {
                        friendList.add(friend);
                }
```

```
                }

        public User get(int id)
        {
                for(int i = 0; i < friendList.size(); i++)
                {
                        if(id == (friendList.get(i)).getId())
                        {
                                return friendList.get(i);
                        }
                }
                return null;
        }

        public ArrayList<User> getAll()
        {
                return friendList;
        }

        public void remove(int id)
        {
                for(int i = 0; i < friendList.size(); i++)
                {
                        if(id == (friendList.get(i)).getId())
                        {
                                friendList.remove(i);
                                break;
                        }
                }
        }
}
```

FriendInvite.java

```
public class FriendInvite implements InviteInterface
{
        private int inviteID;
        private int senderID;
        private String senderName;
        private int receiverID;
        private int type;

        public void setInviteId(int inviteID)
        {
                this.inviteID = inviteID;
        }
        public void setSenderId(int senderID)
        {
                this.senderID = senderID;
        }

        public void setSenderName(String senderName)
        {
                this.senderName = senderName;
        }

        public void setReceiverId(int receiverID)
        {
                this.receiverID = receiverID;
        }

        public void setType(int type)
        {
                this.type = type;
        }

        public int getInviteId()
        {
                return inviteID;
        }
        public int getSenderId()
```

```java
        {
                return senderID;
        }
        public String getSenderName()
        {
                return senderName;
        }

        public int getReceiverId()
        {
                return receiverID;
        }

        public int getType()
        {
                return type;
        }
}
```

InviteCollection.java

```java
import java.util.ArrayList;

public class InviteCollection implements Collection <InviteInterface>
{
        private ArrayList<InviteInterface> inviteList = null;

        public InviteCollection()
        {
                inviteList = new ArrayList<InviteInterface>();
        }

        public void add(InviteInterface invite)
        {
                boolean newInvite = true;
                for(int i = 0; i < inviteList.size(); i++)
                {
                        if(invite.getInviteId() == (inviteList.get(i)).getInviteId())
                        {
                                newInvite = false;
                                break;
                        }
                }
                if(newInvite)
                {
                        inviteList.add(invite);
                }
        }

        public InviteInterface get(int inviteID)
        {
                for(int i = 0; i < inviteList.size(); i++)
                {
                        if(inviteID == (inviteList.get(i)).getInviteId())
                        {
                                return inviteList.get(i);
                        }
                }
                return null;
        }

        public ArrayList<InviteInterface> getAll()
        {
                return inviteList;
        }

        public void remove(int inviteID)
        {
                for(int i = 0; i < inviteList.size(); i++)
                {
                        if(inviteID == (inviteList.get(i)).getInviteId())
```

```
                                                {
                                                        inviteList.remove(i);
                                                }
                                        }
                                }
                        }
}
```

InviteInterface.java

```java
public interface InviteInterface
{
        public void setInviteId(int inviteID);

        public void setType (int type);

        public void setSenderId(int senderID);

        public void setReceiverId(int receiverID);

        public int getInviteId();

        public int getType();

        public int getSenderId();

        public int getReceiverId();
}
```

MainMenuUI.java

```java
/*
* Anon., in Collins, J. (ed.) (2015) 'CS4125: Serenity Gaming – a sample project from Sem1 2014-2015',
* CS4125: Systems Analysis & Design [online],
* available: http://www.csis.ul.ie/coursemodule/CS4125 [accessed 20 Oct 2015]
*/
import javax.swing.*;

import java.awt.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;


public class MainMenuUI extends Menu
{
        private JPanel panel = null;
        private MenuManager menuMgr = null;


        public MainMenuUI()
        {
                showMainMenu();
        }

        public void showMainMenu()
        {
                JPanel mainMenuPanel = new JPanel();
                BorderLayout mainMenuLayout = new BorderLayout();
                mainMenuPanel.setLayout(mainMenuLayout);

                JPanel topBarPanel = new JPanel();
                FlowLayout topBarLayout = new FlowLayout();
                topBarLayout.setAlignment(FlowLayout.RIGHT);
                topBarPanel.setLayout(topBarLayout);
                JLabel welcomeLabel = new JLabel("Welcome " /*+ sessionInfo.getPlayerName()*/);
                topBarPanel.add(welcomeLabel);
                JLabel spacer = new JLabel("          ");
                topBarPanel.add(spacer);
                JButton logoutButton = new JButton("Logout");
                logoutButton.addActionListener(new ActionListener(){
                        @Override
                        public void actionPerformed(ActionEvent e) {
```

```java
                                sessionInfo.logPlayerOut();
                                menuMgr.getMenuFromFactory(1);


                }
        });
        topBarPanel.add(logoutButton);
        mainMenuPanel.add(topBarPanel,mainMenuLayout.NORTH);

        JPanel centerMenuPanel = new JPanel();
        BorderLayout centerMenuLayout = new BorderLayout();
        centerMenuPanel.setLayout(centerMenuLayout);
        centerMenuPanel.setBackground(Color.blue);

        JPanel centerMenuButtonsPanel = new JPanel();

        GridLayout centerMenuButtonsLayout = new GridLayout(3,3);
        centerMenuButtonsPanel.setLayout(centerMenuButtonsLayout);
        JButton gameButton = new JButton("Games");
        gameButton.addActionListener(new ActionListener(){
                @Override
                public void actionPerformed(ActionEvent e)
                {
                        JOptionPane.showMessageDialog(null, "Component not integrated");
                }
        });
        centerMenuButtonsPanel.add(gameButton);
        JButton profileButton = new JButton("Profile");
        profileButton.addActionListener(new ActionListener(){
                @Override
                public void actionPerformed(ActionEvent e) {
                        JOptionPane.showMessageDialog(null, "Component not integrated");
                }
        });
        centerMenuButtonsPanel.add(profileButton);
        JButton friendsButton = new JButton("Friends List");
        friendsButton.addActionListener(new ActionListener(){
                @Override
                public void actionPerformed(ActionEvent e)
                {
                        JOptionPane.showMessageDialog(null, "Component not integrated");
                }
        });
        centerMenuButtonsPanel.add(friendsButton);
        JButton communityButton = new JButton("Communities");
        communityButton.addActionListener(new ActionListener(){
                @Override
                public void actionPerformed(ActionEvent e)
                {
                        JOptionPane.showMessageDialog(null, "Component not integrated");
                }
        });
        centerMenuButtonsPanel.add(communityButton);
        JButton partyButton = new JButton("Party");
        partyButton.addActionListener(new ActionListener(){
                @Override
                public void actionPerformed(ActionEvent e)
                {
                        if(sessionInfo.isPlayerInParty())
                        {
                                menuMgr.getMenuFromFactory(3);
                        }
                        else{
                                int choice = JOptionPane.showConfirmDialog(null, "You are Currently not a
member of a party.\nWould you like to create a new party?", "Create a Party", JOptionPane.YES_NO_OPTION);
                                if(choice == JOptionPane.YES_OPTION)
                                {
                                        System.out.println("clicked yes");
                                        sessionInfo.createParty();
                                }
                        }
```

```
                }
        });
        centerMenuButtonsPanel.add(partyButton);
        centerMenuButtonsPanel.add(communityButton);
        JButton messageButton = new JButton("Messages");
        messageButton.addActionListener(new ActionListener(){
                @Override
                public void actionPerformed(ActionEvent e)
                {
                        menuMgr.getMenuFromFactory(4);
                }
        });
        centerMenuButtonsPanel.add(messageButton);
        centerMenuPanel.add(centerMenuButtonsPanel,centerMenuLayout.CENTER);
        centerMenuPanel.add(centerMenuButtonsPanel,centerMenuLayout.CENTER);

        mainMenuPanel.add(centerMenuPanel,mainMenuLayout.CENTER);

        panel = mainMenuPanel;

    }

    @Override
    public JPanel getMenuPanel()
    {
            return panel;
    }

    @Override
    public void setMenuManager(MenuManager menuMgr)
    {
            this.menuMgr = menuMgr;
    }

}
```

Menu.java

```
/*

* Anon., in Collins, J. (ed.) (2015) 'CS4125: Serenity Gaming – a sample project from Sem1 2014-2015',

* CS4125: Systems Analysis & Design [online],

* available: http://www.csis.ul.ie/coursemodule/CS4125 [accessed 20 Oct 2015]

*/

import javax.swing.*;

public abstract class Menu

{

        protected MenuManager menuMgr;

        protected SessionInformation sessionInfo;

        protected JPanel panel;


        public Menu()

        {


        }

        public abstract JPanel getMenuPanel();

        public void setMenuManager(MenuManager menuMgr)
```

```
                {
                        this.menuMgr = menuMgr;

                }

                public void setSessionInformation(SessionInformation sessionInfo)

                {

                        this.sessionInfo = sessionInfo;

                }

}
```

MenuFactory.java

```
/*
* Anon., in Collins, J. (ed.) (2015) 'CS4125: Serenity Gaming – a sample project from Sem1 2014-2015',
* CS4125: Systems Analysis & Design [online],
* available: http://www.csis.ul.ie/coursemodule/CS4125 [accessed 20 Oct 2015]
*/
import javax.swing.JPanel;

public class MenuFactory
{
        int SCREEN_WIDTH = 0;
        int SCREEN_HEIGHT = 0;

        private SessionInformation sessionInfo = null;

        public MenuFactory()
        {
        }

        public MenuFactory(SessionInformation sessionInfo)
        {
                this.sessionInfo = sessionInfo;
        }

        public Menu getMenu(int menuID,MenuManager menuMgr)
        {
                Menu menu = null;
                 switch(menuID)
                {
                case 1:
                        menu = new StartScreenUI(500,500);
                        menu.setMenuManager(menuMgr);
                        break;
                case 2:
                        menu = new MainMenuUI();
                        menu.setMenuManager(menuMgr);
                        break;
                case 3:
                        menu = new PartyUI();
                        menu.setMenuManager(menuMgr);
                        break;
                case 4:
                        menu = new MessageUI();
                        menu.setMenuManager(menuMgr);
                        break;
                }
                menu.setSessionInformation(sessionInfo);
                menu.setMenuManager(menuMgr);
                return menu;
        }

        public void setScreenWidth(int SCREEN_WIDTH)
        {
                this.SCREEN_WIDTH = SCREEN_WIDTH;
        }
```

```java
        public void setScreenHeight(int SCREEN_HEIGHT)
        {
                this.SCREEN_HEIGHT = SCREEN_HEIGHT;
        }
}
```

MenuManager.java

```java
/*
* Anon., in Collins, J. (ed.) (2015) 'CS4125: Serenity Gaming – a sample project from Sem1 2014-2015',
* CS4125: Systems Analysis & Design [online],
* available: http://www.csis.ul.ie/coursemodule/CS4125 [accessed 20 Oct 2015]
*/
public class MenuManager implements Subject
{
        private MenuFactory menuFac;
        private Screen screen;
        private Menu updatedMenu;

        public MenuManager(MenuFactory menuFac)
        {
                this.menuFac = menuFac;
        }

        @Override
        public void registerObserver(Screen scr)
        {
                screen = scr;
                getMenuFromFactory(1);
                notifyObserver();
        }

        @Override
        public void removeObserver()
        {
                screen = null;
        }

        @Override
        public void notifyObserver()
        {
                screen.update(this);
        }

        public void getMenuFromFactory(int menuID)
        {
                updatedMenu = menuFac.getMenu(menuID,this);
                notifyObserver();
        }

        public Menu getUpdatedMenu()
        {
                return updatedMenu;
        }

        public void setUpdatedMenu(Menu menu)
        {
                updatedMenu = menu;
        }
}
```

Message.java

```java
public interface Message
{
        void setMessageId(int messageID);

        void setSenderId(int senderID);

        void setSenderName(String senderName);
```

```java
        void setReceiverId(int receiverID);

        void setMessage(String message);

        int getMessageId();

        int getSenderId();

        String getSenderName();

        int getReceiverId();

        String getMessage();
}
```

MessageCollection.java

```java
import java.util.ArrayList;

public class MessageCollection implements Collection <Message>
{
        private ArrayList<Message> messageList = null;

        public MessageCollection()
        {
                messageList = new ArrayList<Message>();
        }

        public void add(Message msg)
        {
                boolean newMessage = true;
                for(int i = 0; i < messageList.size(); i++)
                {
                        if(msg.getMessageId() == (messageList.get(i)).getMessageId())
                        {
                                newMessage = false;
                                break;
                        }
                }
                if(newMessage)
                {
                        messageList.add(msg);
                }
        }

        public Message get(int messageID)
        {
                for(int i = 0; i < messageList.size(); i++)
                {
                        if(messageID == (messageList.get(i)).getMessageId())
                        {
                                return messageList.get(i);
                        }
                }
                return null;
        }

        public ArrayList<Message> getAll()
        {
                return messageList;
        }

        public void remove(int messageID)
        {
                for(int i = 0; i < messageList.size(); i++)
                {
                        if(messageID == (messageList.get(i)).getMessageId())
                        {
                                messageList.remove(i);
                        }
```

```
                        }
                }
        }

```

MessageUI.java

/*
* Anon., in Collins, J. (ed.) (2015) 'CS4125: Serenity Gaming – a sample project from Sem1 2014-2015',
* CS4125: Systems Analysis & Design [online],
* available: http://www.csis.ul.ie/coursemodule/CS4125 [accessed 20 Oct 2015]
*/

```java
import javax.swing.*;

import java.awt.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
public class MessageUI extends Menu
{
        private JPanel panel = null;
        private MenuManager menuMgr = null;


        public MessageUI()
        {
                showMessageMenu();
        }

        public void showMessageMenu()
        {
                JPanel mainMenuPanel = new JPanel();
                BorderLayout mainMenuLayout = new BorderLayout();
                mainMenuPanel.setLayout(mainMenuLayout);

                JPanel topBarPanel = new JPanel();
                FlowLayout topBarLayout = new FlowLayout();
                topBarLayout.setAlignment(FlowLayout.RIGHT);
                topBarPanel.setLayout(topBarLayout);
                JLabel uiLabel = new JLabel("Messages");
                topBarPanel.add(uiLabel);
                JLabel spacer = new JLabel("        ");
                topBarPanel.add(spacer);
                JButton logoutButton = new JButton("Logout");
                logoutButton.addActionListener(new ActionListener(){
                        @Override
                        public void actionPerformed(ActionEvent e) {

                                sessionInfo.logPlayerOut();
                                menuMgr.getMenuFromFactory(1);


                        }
                });
                topBarPanel.add(logoutButton);
                mainMenuPanel.add(topBarPanel,mainMenuLayout.NORTH);

                JPanel centerMenuPanel = new JPanel();
                BorderLayout centerMenuLayout = new BorderLayout();
                centerMenuPanel.setLayout(centerMenuLayout);
                centerMenuPanel.setBackground(Color.blue);

                // list of members and leave party button on top row
                JPanel topCenterMenuPanel = new JPanel();
                FlowLayout topCenterMenuLayout = new FlowLayout();
                topCenterMenuLayout.setAlignment(FlowLayout.LEFT);
                topCenterMenuPanel.setLayout(topCenterMenuLayout);
                JList messageList = new JList(); //sessionInfo.getPartyMembers()
                DefaultListSelectionModel m = new DefaultListSelectionModel();
                m.setSelectionMode(ListSelectionModel.SINGLE_SELECTION);
                m.setLeadAnchorNotificationEnabled(false);
                messageList.setSelectionModel(m);
                //messageList.setSelectionMode(ListSelectionModel.SINGLE_SELECTION);
                //messageList.setLayoutOrientation(JList.VERTICAL);
```

```java
JScrollPane messageScroller = new JScrollPane(messageList);
messageScroller.setPreferredSize(new Dimension(300, 150));
topCenterMenuPanel.add(messageScroller);
JPanel messageOptionPanel = new JPanel();
GridLayout messageOptionLayout = new GridLayout(3,0);
messageOptionPanel.setLayout(messageOptionLayout);
JButton createMessageButton = new JButton("Create Message");
createMessageButton.addActionListener(new ActionListener(){
        @Override
        public void actionPerformed(ActionEvent e)
        {
                JOptionPane.showMessageDialog(null, "not implemented yet");
        }
});
messageOptionPanel.add(createMessageButton);
JButton readMessageButton = new JButton("Read Message");
readMessageButton.addActionListener(new ActionListener(){
        @Override
        public void actionPerformed(ActionEvent e)
        {
                JOptionPane.showMessageDialog(null, "not implemented yet");
        }
});
messageOptionPanel.add(readMessageButton);
JButton deleteMessageButton = new JButton("Delete Message");
deleteMessageButton.addActionListener(new ActionListener(){
        @Override
        public void actionPerformed(ActionEvent e)
        {
                JOptionPane.showMessageDialog(null, "Message Deleted.");
        }
});
messageOptionPanel.add(deleteMessageButton);
topCenterMenuPanel.add(messageOptionPanel);
centerMenuPanel.add(topCenterMenuPanel,centerMenuLayout.NORTH);

JPanel midCenterMenuPanel = new JPanel();
FlowLayout midCenterMenuLayout = new FlowLayout();
midCenterMenuLayout.setAlignment(FlowLayout.LEFT);
midCenterMenuPanel.setLayout(midCenterMenuLayout);

JList invitesList = new JList(); //sessionInfo.getPartyMembers()
DefaultListSelectionModel n = new DefaultListSelectionModel();
n.setSelectionMode(ListSelectionModel.SINGLE_SELECTION);
n.setLeadAnchorNotificationEnabled(false);
invitesList.setSelectionModel(n);
//invitesList.setSelectionMode(ListSelectionModel.SINGLE_SELECTION);
//invitesList.setLayoutOrientation(JList.VERTICAL);
JScrollPane inviteScroller = new JScrollPane(invitesList);
inviteScroller.setPreferredSize(new Dimension(300, 150));
midCenterMenuPanel.add(inviteScroller);
JPanel inviteOptionPanel = new JPanel();
GridLayout inviteOptionLayout = new GridLayout(2,0);
inviteOptionPanel.setLayout(inviteOptionLayout);
JButton acceptInviteButton = new JButton("Accept Invite");
acceptInviteButton.addActionListener(new ActionListener(){
        @Override
        public void actionPerformed(ActionEvent e)
        {
                JOptionPane.showMessageDialog(null, "Invite Accepted.\nJoining Party.");
        }
});
inviteOptionPanel.add(acceptInviteButton);
JButton declineInviteButton = new JButton("Decline Invite");
declineInviteButton.addActionListener(new ActionListener(){
        @Override
        public void actionPerformed(ActionEvent e)
        {
                JOptionPane.showMessageDialog(null, "Declined invite.\nInvite removed.");
        }
});
```

```
                        inviteOptionPanel.add(declineInviteButton);
                        midCenterMenuPanel.add(inviteOptionPanel);
                        centerMenuPanel.add(midCenterMenuPanel,centerMenuLayout.CENTER);

                        mainMenuPanel.add(centerMenuPanel,mainMenuLayout.CENTER);

                        JPanel bottomBarPanel = new JPanel();
                        FlowLayout bottomBarLayout = new FlowLayout();
                        bottomBarLayout.setAlignment(FlowLayout.LEFT);
                        bottomBarPanel.setLayout(bottomBarLayout);
                        JButton returnButton = new JButton("<-Return");
                        returnButton.addActionListener(new ActionListener(){
                                    @Override
                                    public void actionPerformed(ActionEvent e) {

                                            //sessionInfo.logPlayerOut();
                                            menuMgr.getMenuFromFactory(2);

                                    }
                        });
                        bottomBarPanel.add(returnButton);
                        mainMenuPanel.add(bottomBarPanel,mainMenuLayout.SOUTH);
                        panel = mainMenuPanel;
            }

            @Override
            public JPanel getMenuPanel()
            {
                        return panel;
            }

            @Override
            public void setMenuManager(MenuManager menuMgr)
            {
                        this.menuMgr = menuMgr;
            }
}
```

Observer.java

```
/*
* Anon., in Collins, J. (ed.) (2015) 'CS4125: Serenity Gaming – a sample project from Sem1 2014-2015',
* CS4125: Systems Analysis & Design [online],
* available: http://www.csis.ul.ie/coursemodule/CS4125 [accessed 20 Oct 2015]
*/
public interface Observer
{
            void update(Subject s);
}
```

Party.java

```
import java.util.ArrayList;

public class Party
{
            private static Party party = null;
            private int partyID;
            private static final int PARTY_SIZE = 5;
            private ArrayList<Integer> partyMemberIDs;

            private Party(int partyID)
            {
                        partyMemberIDs = new ArrayList<Integer>();
                        this.partyID = partyID;
            }

            public static Party getInstance(int partyID)
            {
                        if(party == null)
                        {
```

```java
                    party = new Party(partyID);
            }
            return party;
    }

    public void setMember(ArrayList<Integer> partyMemberIDs)
    {
            this.partyMemberIDs = partyMemberIDs;
    }

    public void removeMember(int id)
    {
            for(int i = 0; i < partyMemberIDs.size(); i++)
            {
                    if(id == partyMemberIDs.get(i))
                    {
                            partyMemberIDs.remove(i);
                    }
            }
            if(this.partyMemberIDs.size() == 0)
            {
                    party = null;
            }
    }

    public int getLeaderId()
    {
            return this.partyMemberIDs.get(0);
    }

    public int getPartyId()
    {
            return partyID;
    }
}
```

PartyInvite.java

```java
public class PartyInvite implements InviteInterface
{
        private int inviteID;
        private int senderID;
        private String senderName;
        private int receiverID;
        private int type;
        private int partyID;

        public void setInviteId(int inviteID)
        {
                this.inviteID = inviteID;
        }
        public void setSenderId(int senderID)
        {
                this.senderID = senderID;
        }

        public void setSenderName(String senderName)
        {
                this.senderName = senderName;
        }

        public void setReceiverId(int receiverID)
        {
                this.receiverID = receiverID;
        }

        public void setType(int type)
        {
                this.type = type;
        }
```

```java
        public void setPartyID(int partyID)
        {
                this.partyID = partyID;
        }

        public int getInviteId()
        {
                return inviteID;
        }
        public int getSenderId()
        {
                return senderID;
        }
        public String getSenderName()
        {
                return senderName;
        }

        public int getReceiverId()
        {
                return receiverID;
        }

        public int getType()
        {
                return type;
        }

        public int getPartyID()
        {
                return partyID;
        }
}
```

PartyUI.java

```java
/*
* Anon., in Collins, J. (ed.) (2015) 'CS4125: Serenity Gaming – a sample project from Sem1 2014-2015',
* CS4125: Systems Analysis & Design [online],
* available: http://www.csis.ul.ie/coursemodule/CS4125 [accessed 20 Oct 2015]
*/
import javax.swing.*;
import java.util.Vector;
import java.awt.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
public class PartyUI extends Menu
{
        private JPanel panel = null;
        private MenuManager menuMgr = null;


        public PartyUI()
        {
                showPartyMenu();
        }

        public void showPartyMenu()
        {
                JPanel mainMenuPanel = new JPanel();
                BorderLayout mainMenuLayout = new BorderLayout();
                mainMenuPanel.setLayout(mainMenuLayout);

                JPanel topBarPanel = new JPanel();
                FlowLayout topBarLayout = new FlowLayout();
                topBarLayout.setAlignment(FlowLayout.RIGHT);
                topBarPanel.setLayout(topBarLayout);
                JLabel uiLabel = new JLabel("Party Menu");
                topBarPanel.add(uiLabel);
                JLabel spacer = new JLabel("        ");
                topBarPanel.add(spacer);
```

```java
JButton logoutButton = new JButton("Logout");
logoutButton.addActionListener(new ActionListener(){
          @Override
          public void actionPerformed(ActionEvent e) {

                    sessionInfo.logPlayerOut();
                    menuMgr.getMenuFromFactory(1);

          }
});
topBarPanel.add(logoutButton);
mainMenuPanel.add(topBarPanel,mainMenuLayout.NORTH);

JPanel centerMenuPanel = new JPanel();
BorderLayout centerMenuLayout = new BorderLayout();
centerMenuPanel.setLayout(centerMenuLayout);
centerMenuPanel.setBackground(Color.blue);
JPanel topCenterMenuPanel = new JPanel();
FlowLayout topCenterMenuLayout = new FlowLayout();
topCenterMenuLayout.setAlignment(FlowLayout.LEFT);
topCenterMenuPanel.setLayout(topCenterMenuLayout);

JList membersList = new JList();
DefaultListSelectionModel m = new DefaultListSelectionModel();
m.setSelectionMode(ListSelectionModel.SINGLE_SELECTION);
m.setLeadAnchorNotificationEnabled(false);
membersList.setSelectionModel(m);
JScrollPane listScroller = new JScrollPane();
listScroller.getViewport().add(membersList);
listScroller.setPreferredSize(new Dimension(300, 150));
topCenterMenuPanel.add(listScroller);
JButton leavePartyButton = new JButton("Leave Party");
leavePartyButton.addActionListener(new ActionListener(){
          @Override
          public void actionPerformed(ActionEvent e)
          {
                    JOptionPane.showMessageDialog(null, "You have left the party");
                    sessionInfo.leaveParty();
                    menuMgr.getMenuFromFactory(2);
          }
});
topCenterMenuPanel.add(leavePartyButton);
centerMenuPanel.add(topCenterMenuPanel,centerMenuLayout.NORTH);

JPanel centerMenuButtonsPanel = new JPanel();

GridLayout centerMenuButtonsLayout = new GridLayout(3,1);
centerMenuButtonsPanel.setLayout(centerMenuButtonsLayout);
JButton inviteButton = new JButton("Invite Friend");
inviteButton.addActionListener(new ActionListener(){
          @Override
          public void actionPerformed(ActionEvent e)
          {
                    JOptionPane.showMessageDialog(null, "Component not integrated");
          }
});
centerMenuButtonsPanel.add(inviteButton);
JButton removeMemberButton = new JButton("Remove Member");
removeMemberButton.addActionListener(new ActionListener(){
          @Override
          public void actionPerformed(ActionEvent e)
          {
                    JOptionPane.showMessageDialog(null, "Component not integrated");
          }
});
centerMenuButtonsPanel.add(removeMemberButton);
JButton gameButton = new JButton("Games");
gameButton.addActionListener(new ActionListener(){
          @Override
          public void actionPerformed(ActionEvent e)
          {
                    JOptionPane.showMessageDialog(null, "Component not integrated");
```

```
                        }
                });
                centerMenuButtonsPanel.add(gameButton);
                centerMenuPanel.add(centerMenuButtonsPanel,centerMenuLayout.CENTER);

                mainMenuPanel.add(centerMenuPanel,mainMenuLayout.CENTER);

                JPanel bottomBarPanel = new JPanel();
                FlowLayout bottomBarLayout = new FlowLayout();
                bottomBarLayout.setAlignment(FlowLayout.LEFT);
                bottomBarPanel.setLayout(bottomBarLayout);
                JButton returnButton = new JButton("<-Return");
                returnButton.addActionListener(new ActionListener(){
                        @Override
                        public void actionPerformed(ActionEvent e)
                        {
                                menuMgr.getMenuFromFactory(2);

                        }
                });
                bottomBarPanel.add(returnButton);
                mainMenuPanel.add(bottomBarPanel,mainMenuLayout.SOUTH);
                panel = mainMenuPanel;
        }

        @Override
        public JPanel getMenuPanel()
        {
                return panel;
        }

        @Override
        public void setMenuManager(MenuManager menuMgr)
        {
                this.menuMgr = menuMgr;
        }
}
```

Player.java

```
/*
* Anon., in Collins, J. (ed.) (2015) 'CS4125: Serenity Gaming – a sample project from Sem1 2014-2015',
* CS4125: Systems Analysis & Design [online],
* available: http://www.csis.ul.ie/coursemodule/CS4125 [accessed 20 Oct 2015]
*/
public class Player extends User
{
        private static Player player = null;
        private Party party;
        private String email;
        private Collection ownedFriendCollection;
        private Collection ownedMessageCollection;
        private Collection ownedInviteCollection;

        private Player()
        {
                ownedFriendCollection = new FriendCollection();
                ownedMessageCollection = new MessageCollection();
                ownedInviteCollection = new InviteCollection();
        }

        public static Player getInstance()
        {
                if(player == null)
                {
                        player = new Player();
                }
                return player;
        }
        @Override
        public int getId()
```

```java
		{
			return id;
		}
		@Override
		public void setId(int id)
		{
			this.id = id;
		}
		@Override
		public String getName()
		{
			return name;
		}
		@Override
		public void setName(String name)
		{
			this.name = name;
		}

		public String getEmail()
		{
			return email;
		}

		public void setEmail(String email)
		{
			this.email = email;
		}

		public void addFriend(User friend)
		{
			ownedFriendCollection.add(friend);
		}

		public void addMessage(Message msg)
		{
			ownedMessageCollection.add(msg);
		}

		public void addInvite(InviteInterface invite)
		{
			ownedInviteCollection.add(invite);
		}

		public void removeFriend(int friendID)
		{
			ownedFriendCollection.remove(friendID);
		}

		public void removeMessage(int MessageID)
		{
			ownedMessageCollection.remove(MessageID);
		}

		public void removeInvite(int inviteID)
		{
			ownedInviteCollection.remove(inviteID);
		}

		public void createParty(int partyID)
		{
			party = Party.getInstance(partyID);
		}
		public int getPartyId()
		{
			return party.getPartyId();
		}
		public boolean isPlayerInParty()
		{
			if(party == null)
			{
```

```java
                                return false;
                        }
                        else
                        {
                                return true;
                        }
                }
                public void removePlayerFromParty()
                {
                        party = null;
                }

                public void resetValues()
                {
                        id = 0;
                        name = "";
                        email = "";
                }
        }
}
```

PrivateMessage.java

```java
public class PrivateMessage implements Message
{
        private int messageID;
        private int senderID;
        private String senderName;
        private int receiverID;
        private String message;

        public void setMessageId(int messageID)
        {
                this.messageID = messageID;
        }

        public void setSenderId(int senderID)
        {
                this.senderID = senderID;
        }

        public void setSenderName(String senderName)
        {
                this.senderName = senderName;
        }

        public void setReceiverId(int receiverID)
        {
                this.receiverID = receiverID;
        }

        public void setMessage(String message)
        {
                this.message = message;
        }

        public int getMessageId()
        {
                return messageID;
        }

        public int getSenderId()
        {
                return senderID;
        }

        public String getSenderName()
        {
                return senderName;
        }

        public int getReceiverId()
```

```
                {
                        return receiverID;
                }

                public String getMessage()
                {
                        return message;
                }
        }
}
```

Screen.java

```
/*
* Anon., in Collins, J. (ed.) (2015) 'CS4125: Serenity Gaming – a sample project from Sem1 2014-2015',
* CS4125: Systems Analysis & Design [online],
* available: http://www.csis.ul.ie/coursemodule/CS4125 [accessed 20 Oct 2015]
*/
import javax.swing.*;

public class Screen extends JFrame implements Observer
{
        private JPanel displayedMenuPanel = null;

        public Screen()
        {
                super("Party maker prototype");
                setSize(500, 500);
                setResizable(false);
                setLocationRelativeTo(null);
                setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        }

        public void update(Subject s)
        {
                dropMenuPanel();

                Menu updatedMenu = s.getUpdatedMenu();

                displayedMenuPanel = updatedMenu.getMenuPanel();

                add(displayedMenuPanel);

                setVisible(true);
        }
        public void dropMenuPanel()
        {
                if(displayedMenuPanel != null)
                {
                        remove(displayedMenuPanel);
                        displayedMenuPanel = null;
                }
        }
}
```

SessionInformation.java

```
/*
* Anon., in Collins, J. (ed.) (2015) 'CS4125: Serenity Gaming – a sample project from Sem1 2014-2015',
* CS4125: Systems Analysis & Design [online],
* available: http://www.csis.ul.ie/coursemodule/CS4125 [accessed 20 Oct 2015]
*/
import java.io.*;
import java.util.Scanner;

import java.util.ArrayList;
import java.util.Vector;
import javax.swing.DefaultListModel;

public class SessionInformation
{
        private static SessionInformation sessionInfo = null;
```

```java
			private DatabaseInterface database;
			private Player player = null;
			private SessionInformation()
			{
					setPlayer();
			}

			public static SessionInformation getInstance()
			{
					if(sessionInfo == null)
					{
							sessionInfo = new SessionInformation();
					}

					return sessionInfo;
			}

			public void setPlayer()
			{
					player = Player.getInstance();
			}

			public void setDbConnection(DatabaseInterface database)
			{
					this.database = database;
			}
			public boolean canUserLogin(String username,String password)
			{
					boolean canLogin = false;
					try
					{
							canLogin = database.canLogin(username, password);
							if(canLogin)
							{
									getPlayerDetails(username);
									getPlayerFriendList();
									getPlayerMessages();
									getPlayerInvites();
							}
					}
					catch(Exception e)
					{
							System.out.println("Error logging in: " + e.toString());
					}
					return canLogin;
			}

			public boolean doesPlayerExist(String username)
			{
					boolean exists = false;
					try
					{
							if(database.doesPlayerExist(username) != 0)
							{
									exists = true;
							}
					}
					catch(Exception e)
					{
							System.out.println("doesPlayerExist Error: " + e.toString());
					}

					return exists;
			}

			public int checkUsernameEmail(String username,String email)
			{
					int existsType = -1;

					try
					{
```

```
                                existsType = database.checkUserNameAndEmail(username, email);
                }
                catch(Exception e)
                {
                                System.out.println("Error validaing username/email: " + e.toString());
                }

                return existsType;
}

public void createPlayer(String username,String password,String email)
{
                try
                {
                                database.createPlayer(username, password, email);
                }
                catch(Exception e)
                {
                                System.out.println("Error creating player: " + e.toString());
                }
}

public void getPlayerDetails(String username)
{
                try
                {
                                String[] details = database.getPlayerDetails(username).split(",");

                                this.player.setId(Integer.parseInt(details[0]));
                                this.player.setName(details[1]);
                                this.player.setEmail(details[2]);
                }
                catch(Exception e)
                {
                                System.out.println("Error getting player details: " + e.toString());
                }
}

public void getPlayerFriendList()
{
                try{
                                ArrayList<String> friends = database.getPlayerFriendList(player.getId());
                                if(friends.size() > 0)
                                {
                                                for(int i = 0; i < friends.size(); i++)
                                                {
                                                                String [] details = (friends.get(i)).split(",");
                                                                User newFriend = new Friend();
                                                                newFriend.setId(Integer.parseInt(details[0]));
                                                                newFriend.setName(details[1]);
                                                                player.addFriend(newFriend);

                                                }
                                }
                }catch(Exception e)
                {}
}

public void getPlayerMessages()
{
                try
                {
                                ArrayList<String> messages = database.getPlayerMessages(player.getId());
                                if(messages.size() > 0)
                                {
                                                for(int i = 0; i < messages.size(); i++)
                                                {
                                                                String [] msgDetails = (messages.get(i)).split(",", 4);
                                                                Message newMessage;
                                                                newMessage = new PrivateMessage();
                                                                newMessage.setMessageId(Integer.parseInt(msgDetails[0]));
                                                                newMessage.setSenderId(Integer.parseInt(msgDetails[1]));
```

```
                                        newMessage.setSenderName(msgDetails[2]);
                                        newMessage.setReceiverId(Integer.parseInt(msgDetails[3]));
                                        newMessage.setMessage(msgDetails[4]);
                                        player.addMessage(newMessage);
                        }
                }
        }catch(Exception e)
        {

        }
}

public void getPlayerInvites()
{
        try
        {
                ArrayList<String> invites = database.getPlayerMessages(player.getId());
                if(invites.size() > 0)
                {
                        for(int i = 0; i < invites.size(); i++)
                        {
                                String [] inviteDetails = (invites.get(i)).split(",");
                                switch(inviteDetails[4])
                                {
                                        case "0":
                                                FriendInvite newFriendInvite = new FriendInvite();

newFriendInvite.setInviteId(Integer.parseInt(inviteDetails[0]));

newFriendInvite.setSenderId(Integer.parseInt(inviteDetails[1]));
                                                newFriendInvite.setSenderName(inviteDetails[2]);

newFriendInvite.setReceiverId(Integer.parseInt(inviteDetails[3]));
                                                newFriendInvite.setType(Integer.parseInt(inviteDetails[4]));
                                                player.addInvite(newFriendInvite);
                                                break;
                                        case "1":
                                                PartyInvite newPartyInvite = new PartyInvite();
                                                newPartyInvite.setInviteId(Integer.parseInt(inviteDetails[0]));

newPartyInvite.setSenderId(Integer.parseInt(inviteDetails[1]));
                                                newPartyInvite.setSenderName(inviteDetails[2]);

newPartyInvite.setReceiverId(Integer.parseInt(inviteDetails[3]));
                                                newPartyInvite.setType(Integer.parseInt(inviteDetails[4]));
                                                newPartyInvite.setPartyID(Integer.parseInt(inviteDetails[5]));
                                                player.addInvite(newPartyInvite);
                                                break;
                                        default:
                                                break;
                                }

                        }
                }
        }catch(Exception e)
        {}
}

public boolean isPlayerInParty()
{
        return player.isPlayerInParty();
}

public void createParty()
{
        try
        {
                int partyID = database.createParty(player.getId());
                player.createParty(partyID);
```

```java
                }catch(Exception e)
                {

                }
        }
        public void leaveParty()
        {
                try
                {
                        int partyID = player.getPartyId();
                        System.out.println(partyID);
                        System.out.println(player.getId());
                        database.removePlayerFromParty(partyID, player.getId());
                        player.removePlayerFromParty();
                }
                catch(Exception e)
                {}
        }

        public String getPlayerName()
        {
                return player.getName();
        }

        public void logPlayerOut()
        {
                if(player.isPlayerInParty())
                {
                        leaveParty();
                }
                System.out.println("logged out");
                player.resetValues();
        }
}
```

StartScreenUI.java

```java
/*
* Anon., in Collins, J. (ed.) (2015) 'CS4125: Serenity Gaming – a sample project from Sem1 2014-2015',
* CS4125: Systems Analysis & Design [online],
* available: http://www.csis.ul.ie/coursemodule/CS4125 [accessed 20 Oct 2015]
*/
import javax.swing.*;
import javax.swing.border.*;

import java.awt.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

public class StartScreenUI extends Menu
{
        private JPanel panel = null;
        private int SCREEN_WIDTH = 0;
        private int SCREEN_HEIGHT = 0;

        public StartScreenUI()
        {
        }

        public StartScreenUI(int SCREEN_WIDTH, int SCREEN_HEIGHT)
        {
                this.SCREEN_WIDTH = SCREEN_WIDTH;
                this.SCREEN_HEIGHT = SCREEN_HEIGHT;
                showLoginScreen();
        }
        public void showLoginScreen()
        {
                final JPanel loginScreen = new JPanel();

                GridBagLayout windowLayout = new GridBagLayout();
                windowLayout.columnWidths = new int[]{SCREEN_WIDTH / 3,SCREEN_WIDTH / 3,SCREEN_WIDTH / 3};
```

```
windowLayout.rowHeights = new int[]{SCREEN_HEIGHT / 3,SCREEN_HEIGHT / 3,SCREEN_HEIGHT / 3};
windowLayout.columnWeights = new double[]{0.0, 0.0, 1.0};
windowLayout.rowWeights = new double[]{0.0, 0.0, 1.0};
loginScreen.setLayout(windowLayout);

GridBagConstraints cons = new GridBagConstraints();
cons.gridx = 1;
cons.gridy = 1;

JPanel loginPanel = new JPanel();

Border margin = new EmptyBorder(10,10,10,10);
Border loginBorder = loginPanel.getBorder();
loginPanel.setBorder(new CompoundBorder(loginBorder,margin));

loginPanel.setBackground(Color.gray);
GridLayout loginLayout = new GridLayout(3,2);
loginLayout.setHgap(10);
loginLayout.setVgap(10);
loginPanel.setLayout(loginLayout);
JLabel usernameLabel = new JLabel("Username:");
loginPanel.add(usernameLabel);
final JTextField getUsername = new JTextField();
loginPanel.add(getUsername);
JLabel passwordLabel = new JLabel("Password:");
loginPanel.add(passwordLabel);
final JPasswordField getPassword = new JPasswordField();
loginPanel.add(getPassword);
JButton loginButton = new JButton("Login");
loginPanel.add(loginButton);
JButton registerButton = new JButton("Register");
loginPanel.add(registerButton);
loginScreen.add(loginPanel,cons);

loginButton.addActionListener(new ActionListener(){
        @Override
        public void actionPerformed(ActionEvent e) {

                String username = getUsername.getText().toLowerCase();
                String password = getPassword.getText();
                if(!username.equals("") && !password.equals(""))
                {
                        boolean login = sessionInfo.canUserLogin(username, password);

                        if(login)
                        {
                                menuMgr.getMenuFromFactory(2);
                                System.out.println("logged in");
                        }
                        else
                        {
                                JOptionPane.showMessageDialog(null,"Invalid login details");
                                getUsername.setText("");
                                getPassword.setText("");
                        }
                }
                else
                {
                        JOptionPane.showMessageDialog(null, "Fields Blank");
                }
        }
});

registerButton.addActionListener(new ActionListener(){
        @Override
        public void actionPerformed(ActionEvent e) {

                loginScreen.setVisible(false);
                showRegisterScreen();
                menuMgr.notifyObserver();
        }
```

```java
                });

                panel = loginScreen;
        }

        public void showRegisterScreen()
        {
                final JPanel registerPanel = new JPanel();

                GridBagLayout windowLayout = new GridBagLayout();
                windowLayout.columnWidths = new int[]{SCREEN_WIDTH / 4, SCREEN_WIDTH / 2, SCREEN_WIDTH / 4};
                windowLayout.rowHeights = new int[]{SCREEN_HEIGHT / 4, SCREEN_HEIGHT / 2, SCREEN_HEIGHT / 4};
                windowLayout.columnWeights = new double[]{0.0, 0.0, 1.0};
                windowLayout.rowWeights = new double[]{0.0, 0.0, 1.0};
                registerPanel.setLayout(windowLayout);

                GridBagConstraints cons = new GridBagConstraints();
                cons.weighty = 1.0;
                cons.weightx = 1.0;
                cons.gridx = 1;
                cons.gridy = 1;
                cons.gridheight = 1;
                cons.gridwidth = 1;

                JPanel registerUserPanel = new JPanel();
                registerUserPanel.setMinimumSize(new Dimension(SCREEN_WIDTH / 2 + 20, SCREEN_HEIGHT / 2));
                Border margin = new EmptyBorder(10,10,10,10);
                Border loginBorder = registerUserPanel.getBorder();
                registerUserPanel.setBorder(new CompoundBorder(loginBorder,margin));
                registerUserPanel.setBackground(Color.lightGray);

                GridLayout registerUserLayout = new GridLayout(7,2);
                registerUserLayout.setHgap(10);
                registerUserLayout.setVgap(10);
                registerUserPanel.setLayout(registerUserLayout);

                final JLabel errorLabel = new JLabel("");
                errorLabel.setForeground(Color.red);
                registerUserPanel.add(errorLabel);
                registerUserPanel.add(new JLabel(""));
                JLabel usernameLabel = new JLabel("Username:");
                registerUserPanel.add(usernameLabel);
                final JTextField getUsername = new JTextField();
                registerUserPanel.add(getUsername);
                JLabel emailLabel = new JLabel("Email:");
                registerUserPanel.add(emailLabel);
                final JTextField getEmail = new JTextField();
                registerUserPanel.add(getEmail);
                JLabel passwordLabel = new JLabel("Password:");
                registerUserPanel.add(passwordLabel);
                final JPasswordField getPassword = new JPasswordField();
                registerUserPanel.add(getPassword);
                JLabel confirmPasswordLabel = new JLabel("Confirm Password:");
                registerUserPanel.add(confirmPasswordLabel);
                final JPasswordField getConfirmPassword = new JPasswordField();
                registerUserPanel.add(getConfirmPassword);
                JButton createButton = new JButton("Create");
                createButton.addActionListener(new ActionListener(){
                        @Override
                        public void actionPerformed(ActionEvent e)
                        {
                                String emailPattern = "^[_A-Za-z0-9-\\+]+(\\.[_A-Za-z0-9-]+)*@"
                                                        + "[A-Za-z0-9-]+(\\.[A-Za-z0-
9]+)*(\\.[A-Za-z]{2,})$";

                                String passwordPattern = "";
                                String username = getUsername.getText().toLowerCase();
                                String email = getEmail.getText().toLowerCase();
                                String password = getPassword.getText();
                                String confirmPassword = getConfirmPassword.getText();

                                if(!username.equals("") && !email.equals("")
```

```java
                                                                && !password.equals("") && !confirmPassword.equals(""))
                                    {
                                            if(password.equals(confirmPassword))
                                            {
                                                    if(email.matches(emailPattern))
                                                    {
                                                            int checkUsernameEmail =
sessionInfo.checkUsernameEmail(username, email);

                                                            if(checkUsernameEmail == 2)
                                                            {
                                                                    sessionInfo.createPlayer(username,
password, email);

                                                                    JOptionPane.showMessageDialog(null,
"User created!");

                                                                    registerPanel.setVisible(false);
                                                                    showLoginScreen();
                                                                    menuMgr.notifyObserver();
                                                            }
                                                            else
                                                            {
                                                                    if(checkUsernameEmail == 0)
                                                                    {

        errorLabel.setText("Username already in use");
                                                                    }
                                                                    else
                                                                    {
                                                                            errorLabel.setText("Email
already in use");
                                                                    }
                                                            }
                                                    }
                                                    else
                                                    {
                                                            errorLabel.setText("Not a valid email address");
                                                    }
                                            }
                                            else
                                            {
                                                    errorLabel.setText("Incorrect Passwords");
                                            }
                                    }
                                    else
                                    {
                                            errorLabel.setText("Field(s) blank");
                                    }
                            }
                    });
                    registerUserPanel.add(createButton);
                    JButton cancelButton = new JButton("Cancel");
                    cancelButton.addActionListener(new ActionListener(){
                            @Override
                            public void actionPerformed(ActionEvent e)
                            {
                                    registerPanel.setVisible(false);
                                    showLoginScreen();
                                    menuMgr.notifyObserver();
                            }
                    });
                    registerUserPanel.add(cancelButton);

                    registerPanel.add(registerUserPanel,cons);
                    panel = registerPanel;
            }
            @Override
            public JPanel getMenuPanel()
            {
                    return this.panel;
            }
}
```

StartUpUI.java

```java
/*
* Anon., in Collins, J. (ed.) (2015) 'CS4125: Serenity Gaming – a sample project from Sem1 2014-2015',
* CS4125: Systems Analysis & Design [online],
* available: http://www.csis.ul.ie/coursemodule/CS4125 [accessed 20 Oct 2015]
*/
import javax.swing.*;
import javax.swing.border.*;

import java.awt.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

public class StartScreenUI extends Menu
{
        private JPanel panel = null;
        private int SCREEN_WIDTH = 0;
        private int SCREEN_HEIGHT = 0;

        public StartScreenUI()
        {
        }

        public StartScreenUI(int SCREEN_WIDTH, int SCREEN_HEIGHT)
        {
                this.SCREEN_WIDTH = SCREEN_WIDTH;
                this.SCREEN_HEIGHT = SCREEN_HEIGHT;
                showLoginScreen();
        }
        public void showLoginScreen()
        {
                final JPanel loginScreen = new JPanel();

                GridBagLayout windowLayout = new GridBagLayout();
                windowLayout.columnWidths = new int[]{SCREEN_WIDTH / 3,SCREEN_WIDTH / 3,SCREEN_WIDTH / 3};
                windowLayout.rowHeights = new int[]{SCREEN_HEIGHT / 3,SCREEN_HEIGHT / 3,SCREEN_HEIGHT / 3};
                windowLayout.columnWeights = new double[]{0.0, 0.0, 1.0};
                windowLayout.rowWeights = new double[]{0.0, 0.0, 1.0};
                loginScreen.setLayout(windowLayout);

                GridBagConstraints cons = new GridBagConstraints();
                cons.gridx = 1;
                cons.gridy = 1;

                JPanel loginPanel = new JPanel();

                Border margin = new EmptyBorder(10,10,10,10);
                Border loginBorder = loginPanel.getBorder();
                loginPanel.setBorder(new CompoundBorder(loginBorder,margin));

                loginPanel.setBackground(Color.gray);
                GridLayout loginLayout = new GridLayout(3,2);
                loginLayout.setHgap(10);
                loginLayout.setVgap(10);
                loginPanel.setLayout(loginLayout);
                JLabel usernameLabel = new JLabel("Username:");
                loginPanel.add(usernameLabel);
                final JTextField getUsername = new JTextField();
                loginPanel.add(getUsername);
                JLabel passwordLabel = new JLabel("Password:");
                loginPanel.add(passwordLabel);
                final JPasswordField getPassword = new JPasswordField();
                loginPanel.add(getPassword);
                JButton loginButton = new JButton("Login");
                loginPanel.add(loginButton);
                JButton registerButton = new JButton("Register");
                loginPanel.add(registerButton);
                loginScreen.add(loginPanel,cons);

                loginButton.addActionListener(new ActionListener(){
```

```java
                @Override
                public void actionPerformed(ActionEvent e) {

                        String username = getUsername.getText().toLowerCase();
                        String password = getPassword.getText();
                        if(!username.equals("") && !password.equals(""))
                        {
                                boolean login = sessionInfo.canUserLogin(username, password);

                                if(login)
                                {
                                        menuMgr.getMenuFromFactory(2);
                                        System.out.println("logged in");
                                }
                                else
                                {
                                        JOptionPane.showMessageDialog(null,"Invalid login details");
                                        getUsername.setText("");
                                        getPassword.setText("");
                                }
                        }
                        else
                        {
                                JOptionPane.showMessageDialog(null, "Fields Blank");
                        }
                }
        });

        registerButton.addActionListener(new ActionListener(){
                @Override
                public void actionPerformed(ActionEvent e) {

                        loginScreen.setVisible(false);
                        showRegisterScreen();
                        menuMgr.notifyObserver();
                }
        });

        panel = loginScreen;
}

public void showRegisterScreen()
{
        final JPanel registerPanel = new JPanel();

        GridBagLayout windowLayout = new GridBagLayout();
        windowLayout.columnWidths = new int[]{SCREEN_WIDTH / 4, SCREEN_WIDTH / 2, SCREEN_WIDTH / 4};
        windowLayout.rowHeights = new int[]{SCREEN_HEIGHT / 4, SCREEN_HEIGHT / 2, SCREEN_HEIGHT / 4};
        windowLayout.columnWeights = new double[]{0.0, 0.0, 1.0};
        windowLayout.rowWeights = new double[]{0.0, 0.0, 1.0};
        registerPanel.setLayout(windowLayout);

        GridBagConstraints cons = new GridBagConstraints();
        cons.weighty = 1.0;
        cons.weightx = 1.0;
        cons.gridx = 1;
        cons.gridy = 1;
        cons.gridheight = 1;
        cons.gridwidth = 1;

        JPanel registerUserPanel = new JPanel();
        registerUserPanel.setMinimumSize(new Dimension(SCREEN_WIDTH / 2 + 20, SCREEN_HEIGHT / 2));
        Border margin = new EmptyBorder(10,10,10,10);
        Border loginBorder = registerUserPanel.getBorder();
        registerUserPanel.setBorder(new CompoundBorder(loginBorder,margin));
        registerUserPanel.setBackground(Color.lightGray);

        GridLayout registerUserLayout = new GridLayout(7,2);
        registerUserLayout.setHgap(10);
        registerUserLayout.setVgap(10);
        registerUserPanel.setLayout(registerUserLayout);
```

```java
                        final JLabel errorLabel = new JLabel("");
                        errorLabel.setForeground(Color.red);
                        registerUserPanel.add(errorLabel);
                        registerUserPanel.add(new JLabel(""));
                        JLabel usernameLabel = new JLabel("Username:");
                        registerUserPanel.add(usernameLabel);
                        final JTextField getUsername = new JTextField();
                        registerUserPanel.add(getUsername);
                        JLabel emailLabel = new JLabel("Email:");
                        registerUserPanel.add(emailLabel);
                        final JTextField getEmail = new JTextField();
                        registerUserPanel.add(getEmail);
                        JLabel passwordLabel = new JLabel("Password:");
                        registerUserPanel.add(passwordLabel);
                        final JPasswordField getPassword = new JPasswordField();
                        registerUserPanel.add(getPassword);
                        JLabel confirmPasswordLabel = new JLabel("Confirm Password:");
                        registerUserPanel.add(confirmPasswordLabel);
                        final JPasswordField getConfirmPassword = new JPasswordField();
                        registerUserPanel.add(getConfirmPassword);
                        JButton createButton = new JButton("Create");
                        createButton.addActionListener(new ActionListener(){
                                @Override
                                public void actionPerformed(ActionEvent e)
                                {
                                        String emailPattern = "^[_A-Za-z0-9-\\+]+(\\.[_A-Za-z0-9-]+)*@"
                                                                            + "[A-Za-z0-9-]+(\\.[A-Za-z0-
9]+)*(\\.[A-Za-z]{2,})$";
                                        String passwordPattern = "";
                                        String username = getUsername.getText().toLowerCase();
                                        String email = getEmail.getText().toLowerCase();
                                        String password = getPassword.getText();
                                        String confirmPassword = getConfirmPassword.getText();

                                        if(!username.equals("") && !email.equals("")
                                                        && !password.equals("") && !confirmPassword.equals(""))
                                        {
                                                if(password.equals(confirmPassword))
                                                {
                                                        if(email.matches(emailPattern))
                                                        {
                                                                int checkUsernameEmail =
sessionInfo.checkUsernameEmail(username, email);
                                                                if(checkUsernameEmail == 2)
                                                                {
                                                                        sessionInfo.createPlayer(username,
password, email);
                                                                        JOptionPane.showMessageDialog(null,
"User created!");
                                                                        registerPanel.setVisible(false);
                                                                        showLoginScreen();
                                                                        menuMgr.notifyObserver();
                                                                }
                                                                else
                                                                {
                                                                        if(checkUsernameEmail == 0)
                                                                        {

        errorLabel.setText("Username already in use");
                                                                        }
                                                                        else
                                                                        {
                                                                                errorLabel.setText("Email
already in use");
                                                                        }
                                                                }
                                                        }
                                                        else
                                                        {
                                                                errorLabel.setText("Not a valid email address");
```

```
                                                        }
                                        }
                                        else
                                        {
                                                errorLabel.setText("Incorrect Passwords");
                                        }
                                }
                                else
                                {
                                        errorLabel.setText("Field(s) blank");
                                }
                        }
                });
                registerUserPanel.add(createButton);
                JButton cancelButton = new JButton("Cancel");
                cancelButton.addActionListener(new ActionListener(){
                        @Override
                        public void actionPerformed(ActionEvent e)
                        {
                                registerPanel.setVisible(false);
                                showLoginScreen();
                                menuMgr.notifyObserver();
                        }
                });
                registerUserPanel.add(cancelButton);

                registerPanel.add(registerUserPanel,cons);
                panel = registerPanel;
        }
        @Override
        public JPanel getMenuPanel()
        {
                return this.panel;
        }
}
```

Subject.java

```
/*
 * Anon., in Collins, J. (ed.) (2015) 'CS4125: Serenity Gaming – a sample project from Sem1 2014-2015',
 * CS4125: Systems Analysis & Design [online],
 * available: http://www.csis.ul.ie/coursemodule/CS4125 [accessed 20 Oct 2015]
 */
public interface Subject
{
        void registerObserver(Screen scr);
        void removeObserver();
        void notifyObserver();
        Menu getUpdatedMenu();
}
```

User.java

```
public abstract class User
{
        protected int id;
        protected String name;


        public abstract int getId();

        public abstract void setId(int id);

        public abstract String getName();

        public abstract void setName(String name);
}
```

TestMain.java

```java
public class TestMain
{
        public static void main(String [] args)
        {
                /* Main: Boots User Interface */
                StartUpUI startUp = new StartUpUI();
                startUp.run();
        }
}
```