Ceng 491 Initial Design Report

Onur Dilek
Ahmet Furkan Üstün
Erkan Yeşilbaş

Frog On Fire

December 19, 2011

Contents

1. Introduction

1.1 Problem Definition

In spite of a deep probe of online game market in order to be able to get sight of an exciting version among famous card games, there is not any extraordinary version of such a classical game, “Pişti”. The possible demand for “Blöflü Pişti” leads us to develop it for mobile platforms. “Blöflü Pişti” will be available and in use for iPhone game market. Because of the fact that the game will use a learning artificial intelligence system, it is going to play better in the process of gaining experience by practicing. Mainly our general goals in this project are:

•To implement a multiplayer mobile game environment.

•To implement a simple but useful serviceable and user-friendly interface.

•To introduce a different version of classical “Pişti” game into mobile game market.

• To integrate AI into the game.

1.2 Purpose

This initial design report intends to provide complete design information and descriptions of “Blöflü Pişti". The aim is the offer a guide to a design that could be easily implemented by any designer reading this report. The end-product will be tested against the requirements to ensure the quality of the software produced.

1.3 Scope

The scope of this document consists of the design patterns of “Blöflü Pişti” project, brief explanation about the goal, objectives and benefits of the project, constraints, assumptions, dependencies, detailed data description and data dictionary, system architecture with its components, user interface and actions of objects, libraries and tools that will be used.

1.4 Overview

This document includes initial design report for “Blöflü Pişti”. At the beginning, an overview of the problem and the product are described. Then, system overview and design considerations are presented to the audience. After declaring the data design of this project, system architecture will be clarified explicitly. Then user interface design and the detailed design of the “Blöflü Pişti” are explained clearly. And finally, time planning of the project will be given by this document.

1.5 Definitions, Acronyms and Abbreviations

BP: Blöflü Pişti

PI: Pisti

AES: Advanced Encryption Standard

SRS: Software Requirements Specification

IDD: Initial Design Report

1.6 References

[1] www.creately.com

[2] http://en.wikipedia.org/wiki/Requirements_traceability

[3] http://itunes.apple.com/us/app/best-card-games-hd/id356111093?mt=8

[4] http://developer.apple.com/devcenter/ios/index.action

[5] http://standards.ieee.org/findstds/standard/1016-1998.html

[6] http://en.wikipedia.org/wiki/Object-relational_mapping

[7] http://aima.cs.berkeley.edu/

[8] http://www.sciencenews.org/sn_arc98/7_18_98/bob1.htm

2. System Overview

"BP" is web-based software which can be used by the registered users and administrators. In order to use the system, users must login with entering their username and passwords which they got when they signed up to system. This user information held by database. The design will be explained more detailed in design considerations and data design sections of this document. The database which holds the user information also starts to keep records of users' all statistics when users sign in to the system. This statistics database is updated with all new records when users signed out and they can see their all updated statistics in their next entrance to the system. The game has two different interfaces, namely user interface and administrator interface. As mentioned before, users will see their all information and statistics in their own interface while administrators can manipulate these statistics with their different interface. The chat system is simple and user-friendly. A review of the current state is done based on game-concept and it is decided that the chat module will stay the same.

3. Design Considerations

3.1 Design Assumptions, Dependencies and Constraints

•**Security:** The communications between the server and client will use WS- Security in Web Services Enhancements for Microsoft .Net (WSE). Secure communication will be provided by XML digital signatures to ensure message integrity and XML encryption for message confidentiality. In order to prevent abuse of the server component, software on the server should block non-ending recurring requests from the same IP address. Server should also log common proxy addresses as well to check if the client is performing DoS attack.

•**Reliability:** User account information will be collected in the database, so if an error were to occur, users should not feel any consequences. There should always be

backups at regular intervals. User statistics will also change how AI agent will play therefore it is of great importance to keep the data consistent.

•**Usability:** Client side of "BP" should be very easy to use with a friendly GUI. As the game is pretty easy so should be the client usage. Users should be able to learn and play the game in minimal time while agents will be of enough challenge for users who wish to improve their game. Also agents can be changed into newcomer "mode" to make it easier for new players.

•**Availability:** All online game applications have the risk of server failure due to excessive number of users. Having regular backups is also important here since it will allow the server to go back to available mode in minimal time. That a server failure was to happen, a reboot and system restoration should commence in a matter of minutes.

Software System Attributes

•**Maintainability:** The modification of the source code should be disabled. The extensions should be applicable directly to the back end, without modification. The patching service should be regarded as a different component in case extensions would be off-loaded to a different server.

•**Portability:** Client component of the software should be portable. Target hardware platform for the client is smart mobile phones namely of Apple's iPhone family. Therefore the client software will be deployed onto iOS operating systems with version 4 and above. However the server side of the game should be independent of specific hardware or software configuration.

•**Scalability:** Most important system attribute of the game should be its scalability. Deploying additional server and dividing the number of players among themselves. And these operations should not require additional extension on the client software.

3.2 Design Goals and Guidelines

The server should support 1000 users concurrently in order for “BP” to be said a massively multiplayer game. Even if any short time delay does not cause to an impediment, it should be better that the delay of communication between server and the user should not exceed 1 second. The database transaction should not consume more than 10% of the time needed to process single request from client. The client software should enable the users to log-on the main menu within 5 seconds. The clients should be able to sign-in in less than 10 seconds. In order for the game to be maintainable, the GUI of the server component should be robust for the administrator to update information or monitor the network traffic. The administrator should be able to log-on to the server within 5 seconds even when the server supports a thousand players online. When the administrator clicks “Manage Accounts” buttons, the GUI should show the requested information within 10 seconds and commit the changes within a minute. Monitoring network traffic should not increase the overhead on server-side as such information can directly be computed on-the-fly. When the administrator clicks monitor network traffic, the GUI on the server component should show the current traffic summary within 10 seconds.

4. Data Design

4.1 Data Description

Data model presented in this section is based on Table and User data objects. Account and AI agent data objects are associated with User data objects. Decks and Hands are associated with tables. Statistics however, keeps information about user's past games and game style. AI agents are not associated directly with any object other than the user that summoned them. Figures in the following sections show and description identify all data objects with their major attributes in high detail.

•***Table:*** This data object represents the area where the users are brought together. Database needs to store the users on the table along with AI agent related to that

user, Deck which will be used to play the game and the cards in the hands of two users.

•***AI_agent:*** The AI agent, as its name implies is an agent of artificial intelligence to play a game against. Beside its unique id, it keeps track of user who summoned it, predefined characteristic information and the table in which it has been placed.

•***Account:*** Account object keeps information related to login and security information. Every user object is associated directly with an account.

•***Deck:*** Deck data object keeps track of available cards in a table. So, instead of preordered set of cards a dynamic method can be used. It also allows the decks to be kept at tables so even if the users leave and rejoin table deck data object stays untouched.

•***User:*** User data object represents the virtual existence of a player. It keeps track of the table in which the player is seated for a game, the AI agents' unique id if any is summoned along with the reference to the account to which it is bound. It also keeps a reference for a Hand object.

•***Hand:*** Hand object keeps references to the cards which are held by a user seated at a table. It is associated directly with a table and then indirectly with a user object, thus it can keep the information relevant to table even when users leave the table.

•***Statistics:*** Statistics are kept at predefined events such as a game ending and/or when a user leave a game before it ends. Its major attributes represent win/loss ratio, total number of games, bluff/actual PI ratio.

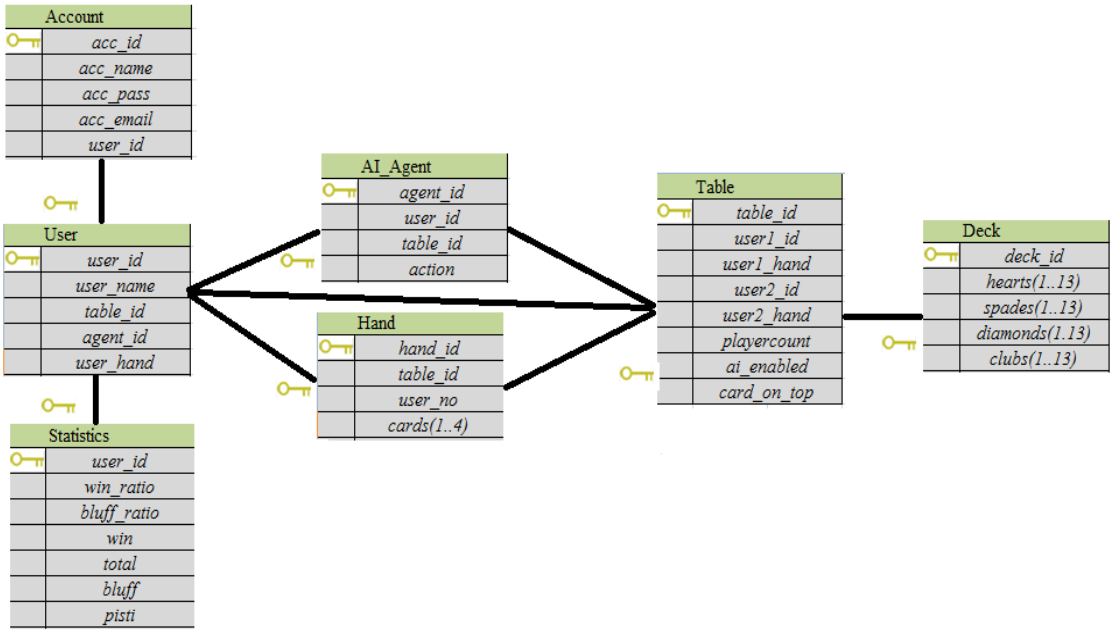

Figure 1: Complete Data Model

4.2 Data Dictionary

This section describes the major system entities; their relative types and descriptions. Attributes of each system entity have been listed.

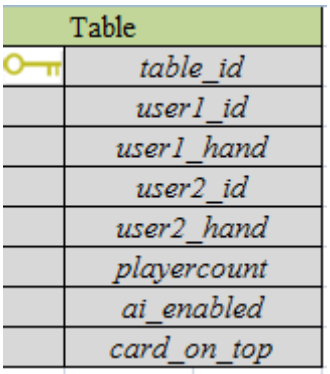

Figure 2: Table Data Object

- ***Table***

table_id: Unique identifier of the table data object.

user1_id: User id of player #1 seated in the table

user1_hand: Reference for the cards held at user #1's hand

user2_id: User id of player #2 seated in the table

user2_hand: Reference for cards held at user #2's hand

playercount: Active number of players (users only) seated at the table.

ai_enabled: Boolean to decide to fill table with an AI agent or not.

card_on_top: The id of the top card (last played). Null if not available.

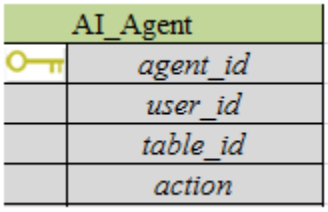

Figure 3: AI_Agent Data Object

•***AI_Agent***

agent_id: Unique identifier of the AI_agent data object.

user_id: User id of the player who summoned the ai agent.

table_id: Table id of the table where agent has been placed.

action: Predefined style of gameplay.

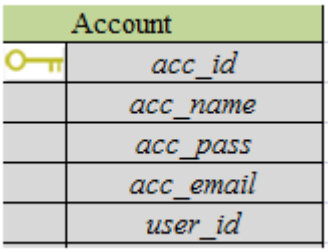

Figure 4: Account Data Object

•***Account***

acc_id:	Unique identifier of the account data object.
acc_name:	Account name specified by the user at registration.
acc_pass:	Password for account security and login
acc_email:	E-mail addresses provided at registration.
user_id:	Associated user identifier.

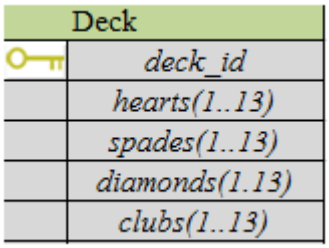

Figure 5: Deck Data Object

•***Deck***

deck_id:	Unique identifier of the deck data object.
hearts(1...13):	Registers for cards belonging to "hearts" series.
spades (1...13):	Registers for cards belonging to "spade" series.
diamonds (1...13):	Registers for cards belonging to "diamonds" series.
clubs (1...13):	Registers for cards belonging to "clubs" series.

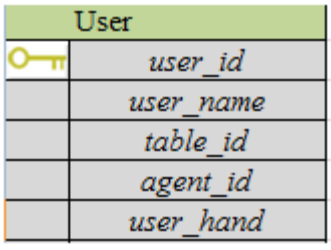

Figure 6: User Data Object

•***User***

user_id: Unique identifier of the user data object.

user_name: Visible user name to use during game.

table_id: Id of table in which user is currently seated.

agent_id: Id of AI agent which user has summoned.

user_hand: Id of hand object which corresponds to this user-table tuple.

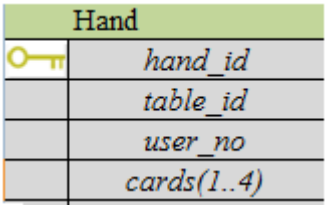

Figure 7: Table Data Object

•***Hand***

hand_id: Unique identifier of the hand data object.

table_id: Table id in which hand is dealt.

user_no: To which user hand is dealt in the table.

cards (1..4): References for cards kept t hand at any time.

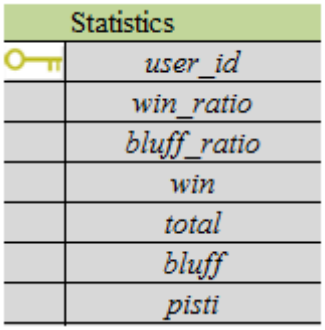

Figure 8: Table Data Object

•***Statistics***

user_id: Unique identifier for statistics as they are also unique per user.

win_ratio: Amount of games won / amounts of games played.

bluff_ratio:	Amount of bluffs versus actual “PI” made.
win:	Number of games won.
total:	Number of games played.
bluff:	Number of times a bluff made.
pisti:	Number of times actual "PI" made.

4.2.1 Relationships

This section explains the relationships between data objects described in the previous section.

Table – User: Every user must be seated at a table in order to play a game. Every table must have at least 1 user seated in it to continue existence. However any user who is not seated in a table has the ability to create a new table or join an existing one.

Table – Deck: Every table owns a deck and every deck must be allocated into a table to be used. A deck cannot be owned by two distinct tables at a time. Deck is created at table creation and is destroyed at table closing.

Table – AI agent: Every agent must be located within a table. However a table can only hold 1 agent (Since having no user causes it to be terminated).

Table – Hand: Every table has 2 Hand object associated with it at creation. Hands are updated at every turn after information regarding played cards is processed.

Table – Statistics: Table requests an update for statistics after special events like "win-lose", "bluff-pisti" and "user leaving". Statistics don't affect table in any way.

User – Account: Each user is associated with at most one account. However an account can delete, modify or have more than one user.

User – AI agent: Every AI agent must be summoned (called) by a user in a table. And every AI agent is created specifically for that user. User may summon or dismiss an AI agent at will.

User – Hand: Every user is associated with a hand when seated in a table. Hands will not be associated with a user only when a table is not full.

User – Statistics: Every user has a statistics object corresponding to it. Although they are paired to keep information, statistics have no effect directly on user. They are only used to keep information related to past games of a user.

AI agent – Hand: Every agent will be summoned into a game therefore will be associated with a Hand object according to their table. AI agent will determine best move according to its hand object. Hand however will not act any different than how it reacts with a user.

AI agent – Statistics: Every agent will be summoned to game by a user. By using that user's id agent collects information from its statistics and if no predefined action is chosen, will adjust its gamestyle according to those information from statistics.

Deck – Hand: Deck creates hands as they get empty. Any card added to Hand will be reduced from the Deck. They will be in constant transfer of card references throughout the game.

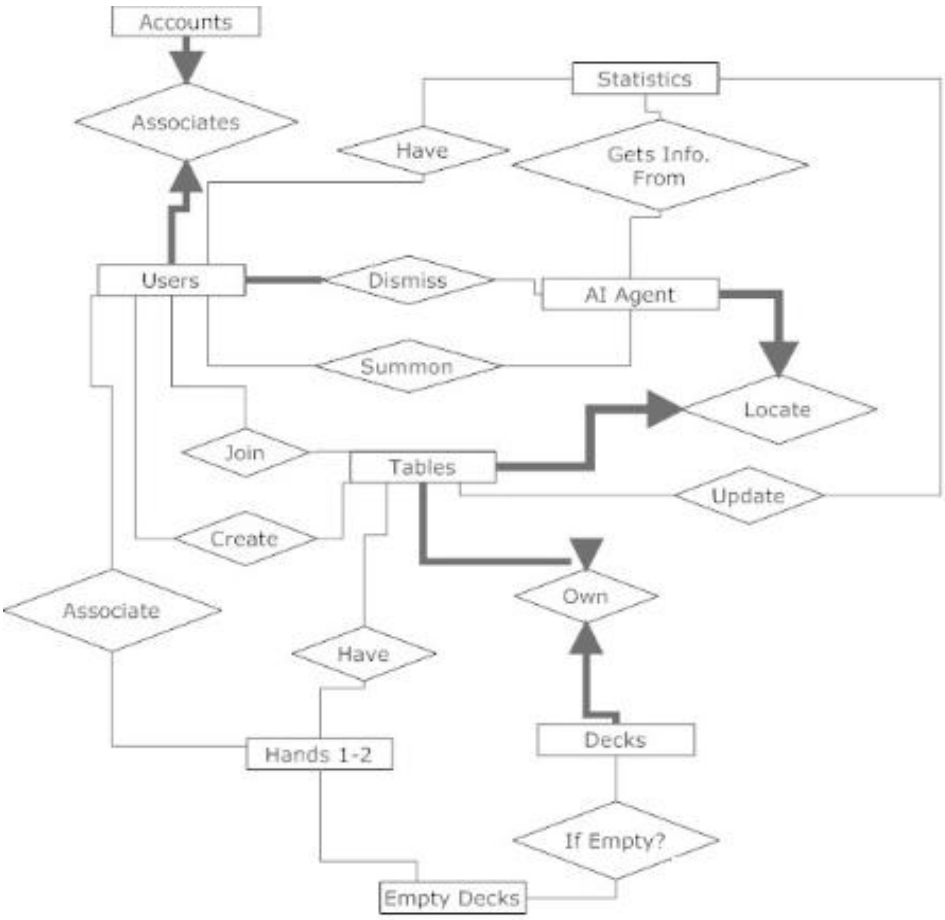

Figure 9: Entity Relationship Diagram

5. System Architecture

5.1 Architectural Design

The game architecture is shown below in the figure. As seen in the figure "BP" has two main components, namely clients and server. The relationship between these two parts is accomplished by package transferring between each part's own communication layers.

Game loop, which belongs to the client's component, interacts with the server component and during this data transfer; information is passed onto server through communication layers. Information can be that of a logging in process or a game interaction where a SQL query will be generated for retrieving relevant information.

The database inside the server will keep record of data such as account information, Table status and user statistics. It will be in constant interaction from various numbers of clients through queries. Following sections will further explain overall architecture of systems and break it down into components and descriptions.

5.2 Description of Components

This section describes the main components of the project. As stated previously two components are present namely, Client and Server. As seen from below figure, the communication between two is carried out through an html .net web service.

5.2.1 Client Network Components

5.2.1.1 Processing Narrative for Client Network Component

Main responsibility of the network is to deliver and receive messages between a client and server tuple. In order to make and initial design some of the possible messages are constructed and listed below. Further detail about any message will be given processing detail of this component.

MessageList::{Register, Login, Logout, TableList, TableCreate, TableJoin, GameStart, AddAgent, Chat, UpdateStats, UpdateDeck, HandDeal, HandRedraw, CardPlay, Bluff };

5.2.1.2 Client Network Component Interface Description

The input interface of the client network component interacts with client graphical user interface. Messages encoded according to transfer protocols and security measurements are transmitted through network. The output interface of the same component however, uses html .Net web service protocols and TCP/IP channels for ultimately reaching the server's network interface.

5.2.1.3 Client Network Component Processing Detail

This section describes the details of how client network component interacts via client user interface and server network interface to allow communications between client-server. Major methods used in this component along with their attributes are introduced below.

Register: This network interaction allow creation of a new player account. A confirmation code and notification will be sent over to user email address as a result of this action. The attributes of this interaction is as given:

```
MessageHead(Client to Server) :: Register
Arguments(Client to Server) :: Username, password,
email, name

MessageHead(Server to Client):: RegisterOK
Arguments(Server to Client):: Username, password

MessageHead(Server to Client):: RegisterFail
Arguments(Server to Client):: Username, password
```

Client to Server		**Server to Client**	
Header	**Arguments**	**Header**	**Arguments**
Register	Username, password, email, name	RegisterOK	Username, password
		RegisterFail	Username, password

Login: This method will allow a player to connect to the game server using a username and a password which have been registered previously. Server will

check username-password tuple from the server database. İf succesfull server will return client_id and user_id else a notificatioın and fault will be returned.

```
MessageHead(Client to Server):: Login
Arguments(Client to Server):: username, password

MessageHead(Server to Client):: LoginOK
Arguments(Server to Client):: Client_id, user_id

MessageHead(Server to Client):: LoginFail
Arguments(Server to Client):: Exception Message
```

Client to Server		**Server to Client**	
Header	**Arguments**	**Header**	**Arguments**
Login	Username, password	LoginOK	Client_ID, user_ID
		LoginFail	Exception message

Logout*:* The client will send his/her user_id and client_id, server will return success or fault to the client.

```
MessageHead(Client to Server):: Logout
Arguments(Client to Server):: Client_id, user_id

MessageHead(Server to Client):: LogoutOK
Arguments(Server to Client):: Success

MessageHead(Server to Client):: LogoutFail
Arguments(Server to Client):: Exception Message
```

Client to Server		**Server to Client**	
Header	**Arguments**	**Header**	**Arguments**
Logout	Client_ID, user_ID	LogoutOK	Success
		LogoutFail	Exception message

Chat: Player sends his/her chat data and it is captured by the server and then displayed to other players. No direct player-to-player communications available.

```
MessageHead(Client to Server):: Chat
Arguments(Client to Server):: ChatData

MessageHead(Server to Client):: Chat
Arguments(Server to Client) :: String ChatData
```

Client to Server		Server to Client	
Header	**Arguments**	**Header**	**Arguments**
Chat	ChatData	Chat	String ChatData

TableList: Player sends a table-list refresh request and server returns currently available tables with given parameters

```
MessageHead(Client to Server):: TableList
Arguments(Client to Server):: Current

MessageHead(Server to Client):: TableListOK
Arguments(Client to Server):: List-of-tables
```

Client to Server		Server to Client	
Header	**Arguments**	**Header**	**Arguments**
TableList	Current	TableListOK	List-of-tables

TableCreate: Player sends a table-creation request and server returns a newly created table information.

```
MessageHead(Client to Server):: TableCreate
Arguments(Client to Server):: None

MessageHead(Server to Client):: TableCreateOK
Arguments(Server to Client):: Table_id
```

Client to Server		Server to Client	
Header	**Arguments**	**Header**	**Arguments**
TableCreate	None	TableCreateOK	Table_ID

TableJoin: Player sends a table id and joining request and the server returns success or fault.

```
MessageHead(Client to Server):: TableJoin
Arguments(Client to Server):: Table_id
```

```
MessageHead(Server to Client):: TableJoinOK
Arguments(Server to Client):: Success

MessageHead(Server to Client):: TableJoinFail
Arguments(Server to Client)::  Exception Message
```

Client to Server		Server to Client	
Header	**Arguments**	**Header**	**Arguments**
TableJoin	Table_ID	TableJoinOk	Success
		TableJoinFail	Exception message

GameStart: Player sends table_id and request to initiate the game. Server returns success or fault.

```
MessageHead(Client to Server):: GameStart
Arguments(Client to Server):: Table_id

MessageHead(Server to Client):: GameStartOK
Arguments(Server to Client):: Success

MessageHead(Server to Client):: GameStartFail
Arguments(Server to Client):: Exception Message
```

Client to Server		Server to Client	
Header	**Arguments**	**Header**	**Arguments**
GameStart	Table_ID	GameStartOK	Success
		GameStartFail	Exception message

AddAgent: Player sends user_id and table_id to request an AI agent to be added to table. Server returns Agent_id or fault.

```
MessageHead(Client to Server):: AddAgent
Arguments(Client to Server):: user_id, table_id

MessageHead(Server to Client):: AddAgentOK
Arguments(Server to Client)::  agent_id

MessageHead(Server to Client):: AddAgentFail
Arguments(Server to Client):: Exception Message
```

Client to Server		Server to Client	
Header	**Arguments**	**Header**	**Arguments**
AddAgent	User_ID, table_ID	AddAgentOK	Agent_ID
		AddAgentFail	Exception message

UpdateStats: Player sends an update request to Server, server updates statistics of the user and returns user_id to check for statistics.

```
MessageHead(Client to Server):: UpdateStats
Arguments(Client to Server):: user_id

MessageHead(Server to Client):: UpdateStatsOK
Argument(Server to Client):: user_id
```

Client to Server		Server to Client	
Header	**Arguments**	**Header**	**Arguments**
UpdateStats	User_ID	UpdateStatsOK	User_ID

UpdateDeck: Following an action on the table user requests Deck to be updated. Server updates the deck and returns deck_id.

```
MessageHead(Client to Server):: UpdateDeck
Arguments(Client to Server):: table_id

MessageHead(Server to Client):: UpdateDeckOK
Argument(Server to Client):: deck_id
```

Client to Server		Server to Client	
Header	**Arguments**	**Header**	**Arguments**
UpdateDeck	Table_ID	UpdateDeckOK	Deck_ID

HandDeal: Following a GameStart player requests the Hands to be filled with cards. Server inserts cards into hands and updates deck accordingly. Returns a new hand object.

```
MessageHead(Client to Server):: HandDeal
Arguments(Client to Server):: table_id, user_no
```

```
MessageHead(Server to Client):: HandDealOK
Arguments(Server to Client):: hand_id
```

Client to Server		Server to Client	
Header	**Arguments**	**Header**	**Arguments**
HandDeal	Table_ID, user_no	HandDealOK	Hand_ID

HandRedraw: When hand of a player gets empty it requests to be refilled. Server fills hand and returns the hand_id. Updates deck on the background accordingly.

```
MessageHead(Client to Server):: HandRedraw
Arguments(Client to Server):: hand_id

MessageHead(Server to Client):: HandRedrawOK
Arguments(Server to Client):: hand_id
```

Client to Server		Server to Client	
Header	**Arguments**	**Header**	**Arguments**
HandRedraw	Hand_ID	HandRedrawOK	Hand_ID

CardPlay: Player chooses a card to play from hand. Server checks top card and returns match_event if cards match success if not match.

```
MessageHead(Client to Server):: CardPlay
Arguments(Client to Server):: cards(1..4), hand_id

MessageHead(Server to Client):: CardPlayOK
Arguments(Server to Client):: success, card_on_top

MessageHead(Server to Client):: CardPlayMatch
Arguments(Server to Client):: Match_event,
card_on_top <= Null
```

Client to Server		Server to Client	
Header	**Arguments**	**Header**	**Arguments**
CardPlay	Cards(1..4), hand_ID	CardPlayOK	Success, card_on_top
		CardPlayMatch	Match_event, card_on_top <= Null

Bluff: Player chooses a card from hand to play face down. Server asks other player whether to accept or challange the bluff. Then return the result of bluff.

```
MessageHead(Client to Server):: Bluff
Arguments(Client to Server):: cards(1..4), hand_id

MessageHead(Server to Client):: BluffPop
Arguments(Server to Client):: None

MessageHead(Server to Client):: BluffResult
Arguments(Server to Client):: None
```

Client to Server		**Server to Client**	
Header	**Arguments**	**Header**	**Arguments**
Bluff	Cards(1..4), hand_ID	BluffPop	None
		BluffResult	None

5.2.1.4 Dynamic Behaviour of Client Network Component

The behavior of client network component can be explained as this: Player sends a request to Server with the following behavior.

• Player --- Client Network Component --- Server Network Component ---- Server

• Request ==> Send Request Message ==> Translate Message

• Translate Response <== Send Request Response <== Respond Request

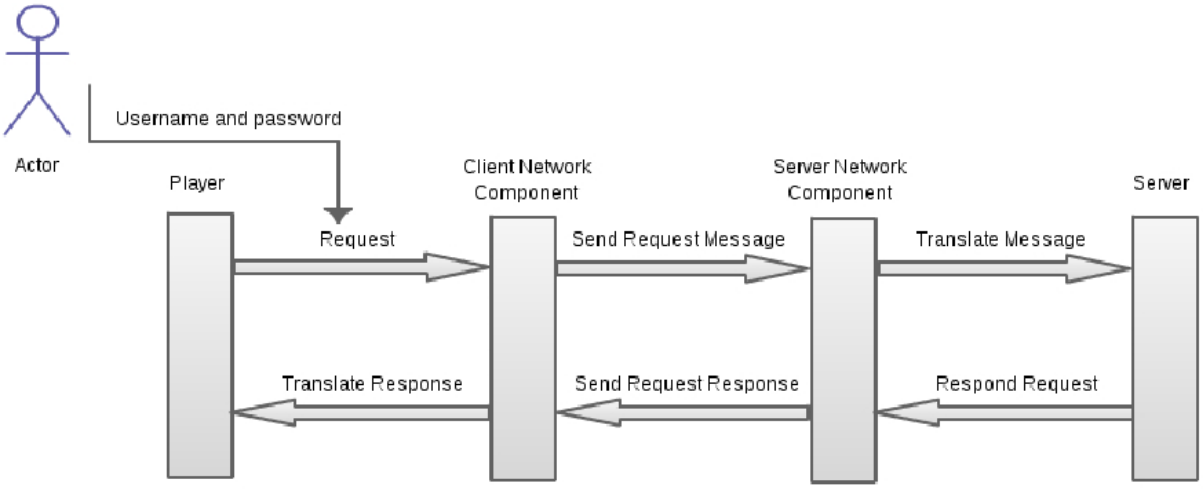

Figure 10: Sequence Diagram for Client Network Component

5.2.2 Server Network Component

5.2.2.1 Processing Narrative for Server Network Component

There are numerous messages that are sent to client according to their requests. A list of these messages is shown below.

```
MessageList::{RegisterOK,RegisterFail,LoginOK,
LoginFail,Chat,TableListOK,TableCreateOK,TableJoinOK,
TableJoinFail,GameStartOK, GameStartFail,AddAgentOK,
AddAgentFail,UpdateStatsOK, UpdateDeckOK, HandDealOK,
HandRedrawOK, CardPlayOK, CardPlayMatch, BluffPop,
BluffResult}
```

5.2.2.2 Server Network Component Interface Description

The server network component interfaces with all client network components simultaneously with half-duplex manner. The transmission between is html .Net web service implementation over TCP.

5.2.2.3 Server Network Component Processing Detail

The most important attribute of the server network component is that it prioritizes requests and thus allows for server to run smoothly without being overwhelmed by requests. Since game is turn based and not real time, a slight latency between requests and their responses does not effect gameplay so long as they don't exceed our design goals in part 3.2. So we can give a pseudo-algorithm for priority query formed by server network component as given:

- Gather requests from clients.
- Prioritize according to action type,
- Prioritize according again by action time among same time of actions.
- Send most urgent set of messages to server.
- If query is empty go back to 1 else go back to 4.

5.2.2.4 Dynamic Behaviour of Server Network Component

The behavior of server network component has been explained below.

Player sends a request to Server with the following behavior.

•Player --- Client Network Component --- Server Network Component ---- Server

•Request ==> Send Request Message ==> Translate Message

•Translate Response <== Send Request Response <== Respond Request

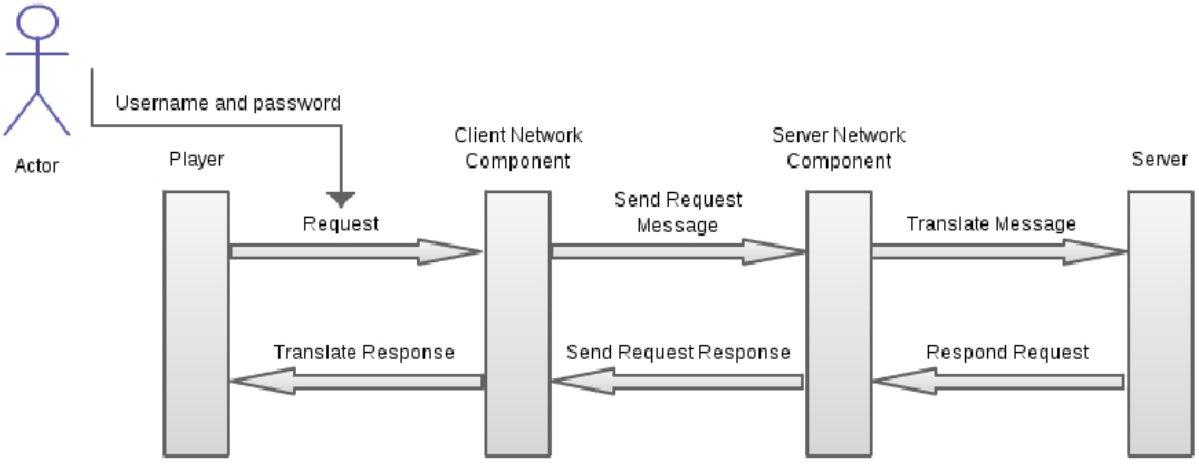

Figure 11: Sequence Diagram for Server Network Component

5.2.3 Management Component

The management components will provide utilities that only administrators need. It encapsulates few crucial processing units of the game such as "data management" and "game loop". With a proper design with object oriented approach, each subcomponent will enable single and consistent interface to other components of the system.

5.2.3.1 Processing Narrative for Management Component

This component is located on server side. Major sub-components are Account management, virtual environment management and game management. This component is aimed to be used by system administrators. Descriptions and processing details for each sub-component is given in the following sections of this document.

5.2.3.2 Management Component Interface Description

The component interfaces with administrators via a GUI. Any input entered by administrators will be handled directly within this component. The results however may and will be transmitted to other components when necessary.

5.2.3.3 Management Component Processing Detail

The processing functions for each subcomponent are given in this section. Following paragraphs describe the detailed description of the subcomponents.

Virtual Environment Management

The administrator can use this component to edit virtual environment. Virtual environment consists of Tables and rooms filled with users. Any change made will be reflected to clients using the network component of the server.

Account Management

Accounts of players can be managed through user interface of this management component. Tasks which include deleting, inserting account and manipulating account information can be done here.

Game Management

This management component will be used to debug or change game mechanics if need arise or for special occasions.

5.2.3.4 Dynamic Behaviour of Management Component

The dynamic behavior of the management component can be explained as:

Administrator sends a management request to Server with the following behavior.

•Administrator --- Management Component ---- Server

• Management Request ==> Transmit Message

• Update GUI <== Respond Request

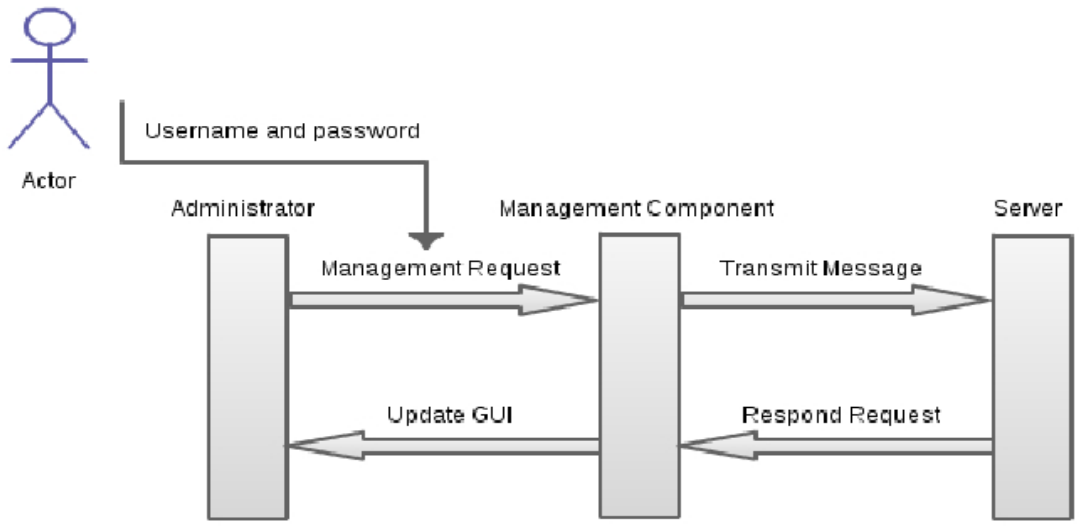

Figure 12: Sequence Diagram for Management Component

5.3 Design Rationale

The decomposition of the game and above components is agreed upon by all members. Most problematic discussion topic was whether to allow actions prior to server approval. It allowed for lower response time for client and reduced server load. However it reduced 1 layer of extra protection against cheating. But since this is not our only "anti-cheat" component in our design we agreed on said components.

Another point we discussed deeply was server-client network architecture over with the other group in the past weeks. But now it is decided as we agreed that a simple client-server architecture with up to 128 bit EAS with predefined key for security.

5.4 Traceability of Requirements

The relation between the functions in Software Requirements Specification report and methods described in this document is listed below. The table shows the interaction between users' functions and IDD methods

Users' Functions	IDD Methods
Sign-up	Register
Sign-in	Login
Enter to a Table	TableJoin
New Table	TableCreate
Wait For Rival	TableCreate
Play Against AI Agent	AddAgent
Existing Table	TableJoin
Watch Game	TableJoin
Play Against Rival	TableJoin
See the Statistics	Logout

Figure 13: Traceability Table

6. User Interface Design

6.1 Overview of User Interface

The interaction between user and game is accomplished using the user interface. After opening the application from iPhone's menu, user waits for application to load and connect to the server. Then prompted to chose log in or register, followed by filling out log in information. If authentication is successful player is located into the "lobby" where he can create a new table or join an existing table. User uses touch-screen capabilities of iPhone and even uses its on-screen keyboard. When the user quits the game, interaction between client and server ends. Then program shuts down.

6.2 Screen Images

• **Account Screen:** The first step when the user runs the game is this screen. If the user has already had an account, he/she can login with "Sign in" button using that information.

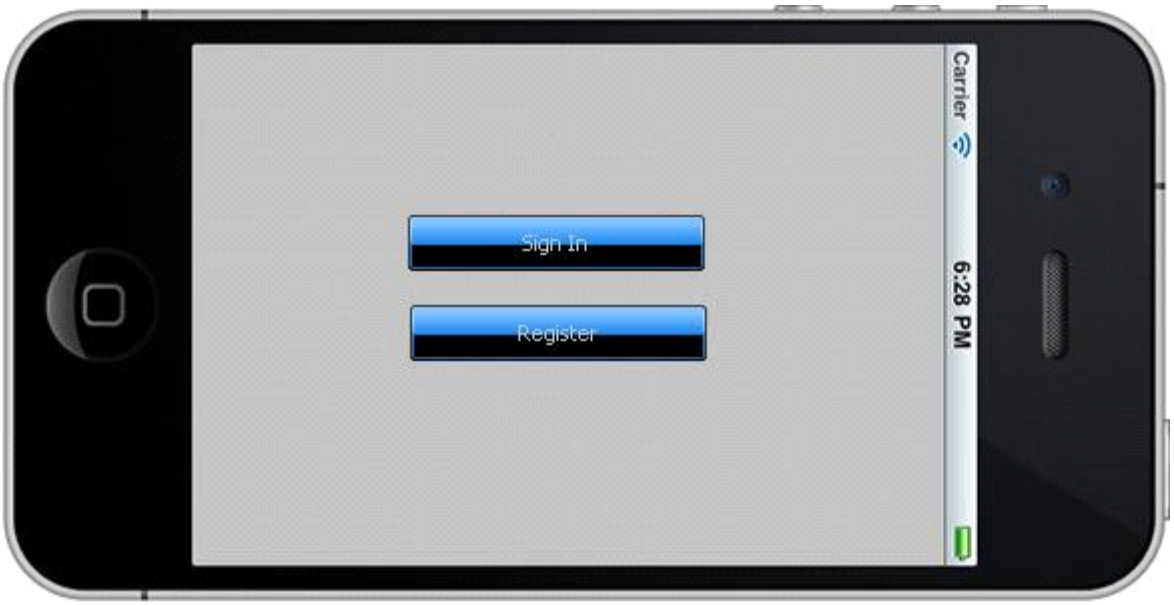

Figure 14: Account Screen

• **Create Account Screen:** In order to create a new account he/she selects "register" button

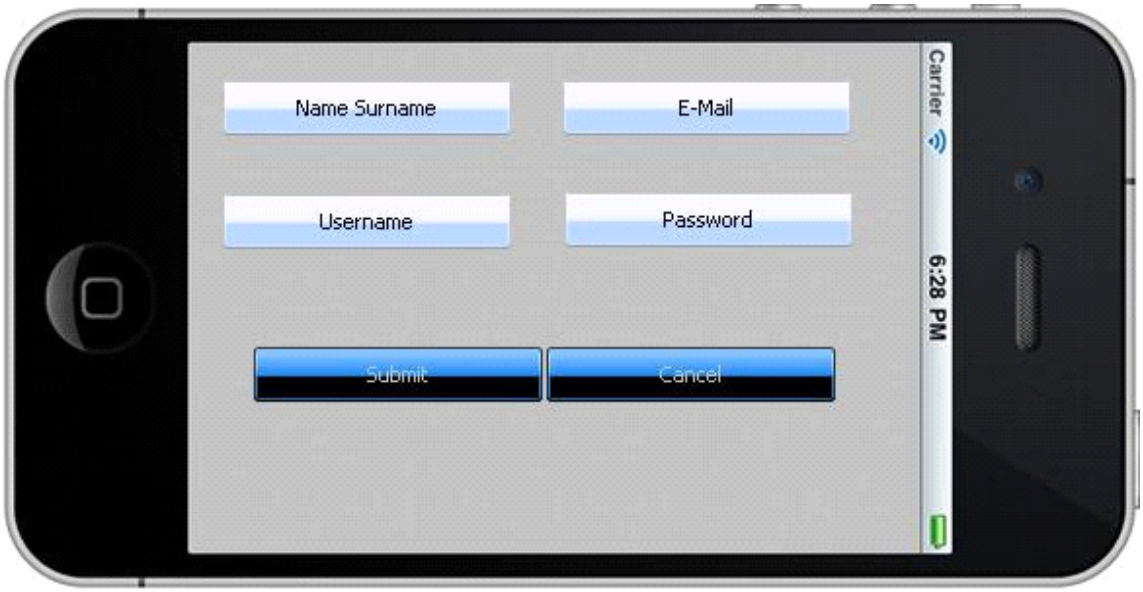

Figure 15: Create Account Screen

• **Lobby Screen:** This is the main display area. There is some information about player points. There are also buttons to create a new table and join an existing table in this

screen.

Figure 16: Lobby Screen

•**Table Screen:** This is the area where actual game happens. After 2 players are seated at a table game can begin. Each player is dealt 4 cards and then expected to play 1 till they use all 4 cards then they will get 4 new cards and it will continue like that until deck is empty. There is also "Add AI" button in this display so that a user can play against and AI agent.

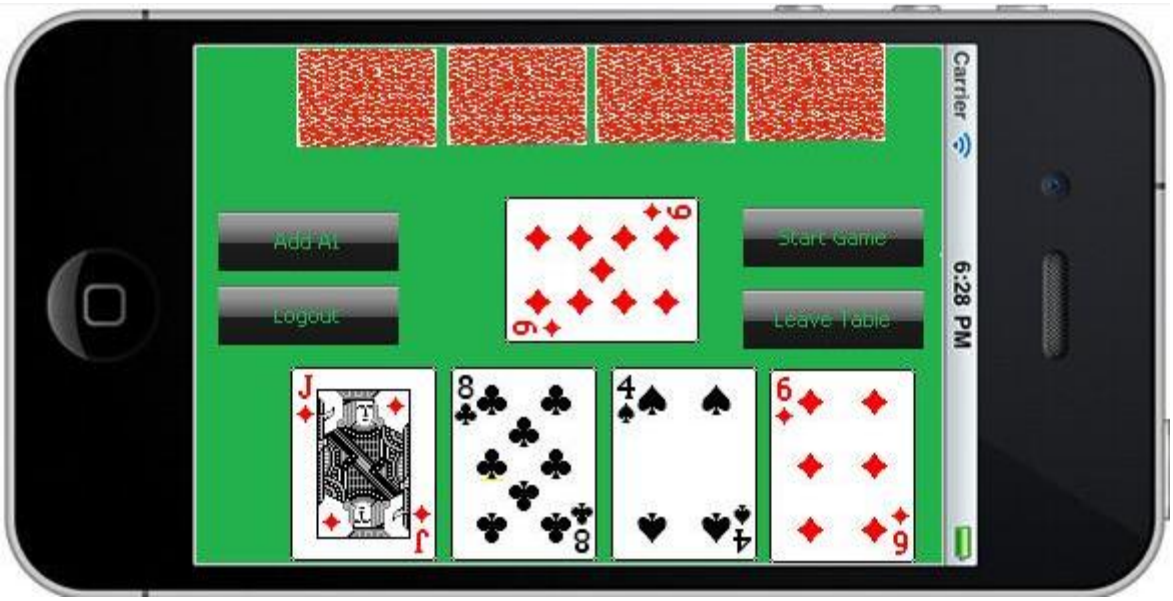

Figure 17: Table Screen

•**Statistics View:** This screen comes whenever "MyStats" button is pressed at a table. It displays a characters own statistics along with other information about the user.

Figure 18: Statistics View Screen

•**Logout Screen:** Whenever "logout" button has been pressed it will show a popup prompt button with options "ok" and "cancel" whenever logout is confirmed by pressing "ok" button game will end.

Figure 19: Logout Screen

6.3 Screen Objects and Actions

When the user executes the game first screen that comes across will be account page. Where user can choose to register or log in. Once logged into the system player is placed into the lobby to see other games. One can however choose to create his/her own table and then wait for opposition. Once inside a table and the table gets full game can commence. However player can at any time choose to leave the game by pressing "logout" button and then confirming the action by pressing "ok" popup button. This flow can be traced in the following Figure game flow.

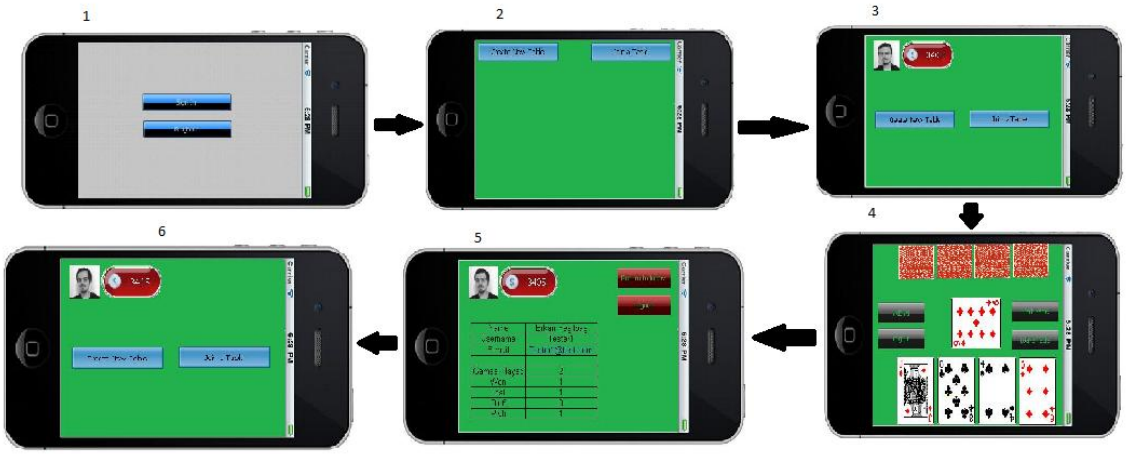

Figure 20: Game Screen Flowchart

7. Libraries and Tools

Client will be an application of Apple's iPhone, thus will be using iOS operating system. Server will be Windows NT family operating system. As both components will be developed with C like languages namely Objective-C and C#. Microsoft's .Net framework will be used for server side. XNA 4.0 runtime libraries will be available to use for client side graphics computation but since no 3d graphics computations are required, no physics engine will be required. For network communication, the client and the server software will depend html .Net webservice libraries.

The interaction between server and client will be maintained over TCP channel. All other communications will be carried out on shared memory.

8. Time Planning

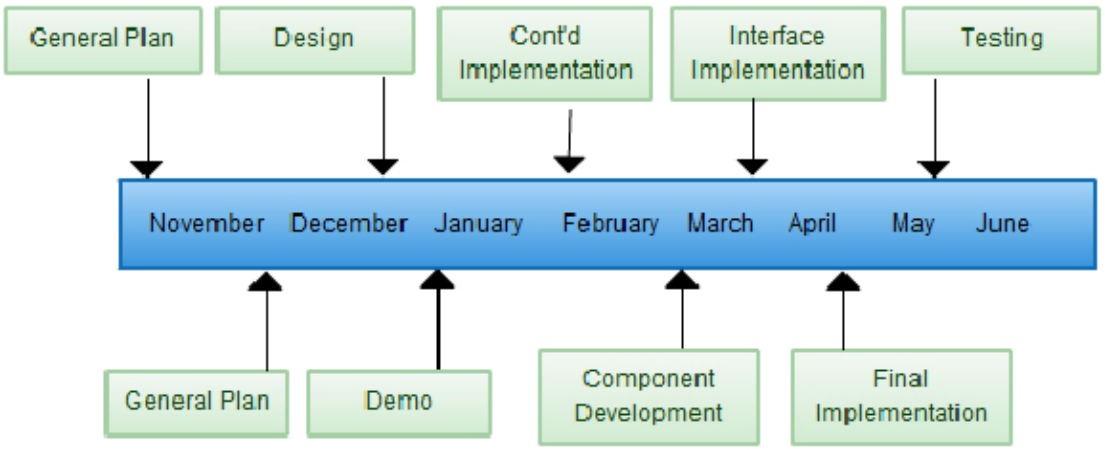

Figure 21 Estimation Diagram[1]

8.1 Term 1 Gannt Chart

Number	Task	Start	End	Duration	2011			2012
					October	November	December	January
1	**Project Management**	14.10.201	28.05.2012	216				
2	Proposal Report	20.10.2011	31.10.2011	11				
3	SRS Report	09.11.2011	25.11.2011	16				
4	IDD Report	05.12.2011	20.12.2011	15				
5	Detailed Design Report	01.01.2012	15.01.2012	14				
6	Prototype Demo	16.01.2012	28.01.2012	12				
7	**System Component Analysis**	10.11.2011	16.01.2011	67				
8	Unit Tests	10.11.2011	01.12.2011	21				
9	Interface Analysis	30.11.2011	31.12.2011	30				
10	Server-Client Analysis	01.12.2011	16.01.2012	47				
11	SQL Server Configuration	01.12.2011	05.01.2012	22				
12	Server Software Development	14.12.2011	10.01.2012	27				
13	AI Analysis	01.01.2012	16.01.2012	15				
14	Unit Tests #2	05.01.2012	10.01.2012	5				
15	Social Interaction Configuration	09.01.2012	17.01.2012	8				

Figure 22: Term 1 Gannt Chart

8.2 Term 2 Gannt Chart

Number	Task	Start	End	Duration	2012			
					February	March	April	May
1	**System Configuration**	15.02.2012	01.04.2012	44				
2	Server Configuration	15.02.2012	21.02.2012	7				
3	Network Traffic Monitoring	27.02.2012	10.03.2012	14				
4	Database Optimization	09.03.2012	15.03.2012	7				
5	AI Configuration	20.03.2012	01.04.2012	11				
6	**System Design Development**	01.04.2012	25.04.2012	24				
7	Client GUI Design	01.04.2012	04.04.2012	3				
8	Server GUI Design	01.04.2012	04.04.2012	3				
9	Scenario & Character Design	06.04.2012	09.04.2012	3				
10	System Resource Development	09.04.2012	14.04.2012	5				
11	Sound Integration	19.04.2012	22.04.2012	4				
12	**System Testing**	25.04.2012	20.05.2012	25				
13	Server-Client Test	25.04.2012	02.05.2012	8				
14	Network Traffic Test	01.05.2012	06.05.2012	7				
15	Database Test	07.05.2012	10.05.2012	4				
16	AI Test	11.05.2012	14.05.2012	4				
17	Alpha Test	13.05.2012	15.05.2012	2				
18	Beta Test & Debugging	16.05.2012	20.05.2012	5				
19	Finalization	21.05.2012	28.05.2012	7				
20	Post Production	21.05.2012	28.05.2012	8				

Figure 23: Term 2 Gannt Chart

9. Conclusion

This Initial Design Report was prepared to create a bridge between the design and implementation parts of the project. This document specifies the brief software design specifications, which are organized according to the specifications in IEEE Recommended Practice 1016-1998, for “BP”. This report includes data flow models, ER diagram,interface features, estimation diagram and Ganntt Charts. The possible design and other constraints that can be faced are explained. The tools and the libraries that will be used to develop the project are given. A brief detailed design explained. We have explained the works that we have done so far and we give the time planning that will be followed by group members.