2015

MIDDLE EAST TECHNICAL UNIVERSITY

DEPARTMENT OF COMPUTER ENGINEERING

# *Poker Playing Agent*

## *Software Design Description*

REVISION: 1.0

STANDARD: IEEE SDD 1016

A4

HÜSEYIN ALEÇAKIR

HÜSEYİN AYDIN

HEVAL AZİZOĞLU

ZÜLFİYE ARĞIN

# PREFACE

This document contains the system design information for Poker Playing Agent Project. The document is prepared according to the “IEEE Standard for Information Technology Systems Design –Software Design Descriptions – IEEE Std. 1016 – 2009”.

This Software Design Documentation provides a complete description of all the system design and views of Poker Playing Agent Project.

The first section of this document presents scope, purpose, overview, intended audience, reference material and definitions and abbreviations of the project.

The second chapter includes definitions which are used in the document.

The third chapter describes the conceptual model for software design descriptions.

The fourth chapter of this document specifies the design description content for the whole system.

The last chapter of this document includes design viewpoint of the Poker Playing Agent Project.

## Record of Changes

| Version Number | Date | Number of Section | A – Added<br>M – Modified<br>D - Deleted | Title or brief description |
|---|---|---|---|---|
| 1.0 | 04.01.2015 | | | First Release |
| 1.1 | 01.03.2015 | 4.7 | M | Motivation of project was added |
| 1.1 | 01.03.2015 | 5.2 | M | Usage of the system was divided according to user type |
| 1.1 | 01.03.2015 | 5.3 | M | The explanation of components was added |
| 1.1 | 01.03.2015 | 5.4 | M | Game API was included |
| 1.1 | 01.03.2015 | 5.6 | M | Cardinality constraints were added |
| 1.1 | 01.03.2015 | 5.7 | M | Observer Pattern was removed |
| 1.1 | 01.03.2015 | 5.11 | M | Bayesian Network diagram was added |

# Table of Contents

# 1 OVERVIEW

## 1.1 SCOPE

In this document, design description for the Poker Playing Agent project will be provided to the audience. A structural overview of all modules, data and interfaces will be defined. There will be a set of design views to be more specific about the design and development process of the project. Overall, this documentation will give a basic understanding of how the project will be implemented. As a base reference, Software Requirements Specification of this project will be used.

## 1.2 PURPOSE

The main purpose of this document is to show and detail the architecture of Poker Playing Agent Project with the help of design viewpoints. This document will be used as a reference by the development team in implementation phase. In order to specify the design and development processes and supply a simple and basic architecture, a group of design views will be offered with UML diagrams. Also an explicit understanding about how the project will be implemented will be provided with the algorithmic explanations. Contents of this document may include some assumptions which will have certainty after implementation.

## 1.3 INTENDED AUDIENCE

This document is intended for various types of stakeholders such as the developers of the PPA system, forthcoming researchers of Computer Poker domain and game designers and developers in the game industry that may be willing to use agent developed within this project to provide a guidance to them.

## 1.4 REFERENCES


- IEEE. IEEE Std 1016-2009 IEEE Recommended Practice for Software Design Description.. IEEE Computer Society, 2009.

- StarUML 5.0 User Guide. (2005). Retrieved from http://staruml.sourceforge.net/docs/user- guide(en)/toc.html

- “Poker Playing Agents” Software Specification Requirements (SRS) Document, 2014.

- “Opponent Modeling in Reinforcement Learning for Partially Observable Stochastic Games: Poker, a case study”, AI & R Lab, 2014.

- “Bayesian Networks” in Ruggeri F., Faltin F. & Kenett R., Encyclopedia of Statistics in Quality & Reliability, Wiley & Sons, Ben-Gal I., 2007.


# 2 DEFINITIONS, ACRONYMS AND ABBREVIATIONS

| ABBREVIATION | DEFINITION |
|---|---|
| SDD | Software Design Description |
| SRS | Software Requirements Specification |
| UC | Use Case |
| UML | Unified Markup Language |
| IDE | Integrated Development Environment |
| PPA | Poker Playing Agent |
| BN | Bayesian Network |
| PDT | Probabilistic Decision Tree |

# 3 CONCEPTUAL MODEL FOR SOFTWARE DESIGN DESCRIPTIONS

In this section, a conceptual model for SDD will be given. Conceptual model contains basic terminology and concepts of the document. Furthermore, the context of SDD will be explained with this conceptual model.

## 3.1 SOFTWARE DESIGN IN CONTEXT

### 3.1.1 Application Overview

The main aim of Poker Playing Agents project is to produce computer poker agents which have ability to beat average human players and well-known computer poker agents in the literature. This project will provide an efficient and effective agent to Computer Poker literature which will be implemented with the state of the art methodology. Also with this project, team will introduce a testing methodology to evaluate agents' performance and decisions. Users which encapsulate other developers and researchers can adapt and test their bots with this system.

In general, PPA system consists of two major components; agent component and third party poker game client. Agent component will include two subsystems; learning part and decision part. These two subsystems will work collaboratively. Any operation done in learning part will affect decision part. Additional information to decide next action of poker bot will be supplied by learning part. The other component, game client, will provide user interfaces and testing environment to the system. Poker Academy is chosen as this third party application for now.

### 3.1.2 Technologies Used

Poker Playing Agent system will be developed using Java programming language on NetBeans IDE as development environment. System requires agents to be compiled with JDK 5 to adapt their class files to game client. System will use MySQL database server for database purposes. StarUML will be used in the design process to create required diagrams.

## 3.2 SOFTWARE DESIGN DESCRIPTIONS WITHIN THE LIFE CYCLE

### 3.2.1 Influences on SDD Preparation

The software requirements specification (SRS) for Poker Playing Agent project is the key software life cycle product that drives software design. Software requirements specification includes functional and non-functional software requirements that drive design process and also design constraints that should be considered and observed during design.

### 3.2.2 Influences on Software Life Cycle Products

Some of the requirements may change during the preparation stage of SDD and/or during the implementation phase of the project. Also, SDD may affect test plans and test documentation of Poker Playing Agent Project.

### 3.2.3 Design Verification and Design Role in Validation

Design shall be verified and validated during the testing stage. System parts will be checked with the prepared test cases. With these test cases, it will be determined whether the system fulfills the specified requirements or not. In validation progress, the system will be tested whether it is the correct system. In the verification progress, the system will be tested whether it is being developed correctly.

# 4 DESIGN DESCRIPTION INFORMATION CONTENT

## 4.1 INTRODUCTION

This SDD document is prepared for Poker Playing Agent project. It describes how this system will be designed and implemented. Detailed design cases will be explained in the following sections by using necessary diagrams, design views and design viewpoints about the structure of the PPA.

## 4.2 SDD IDENTIFICATION

After testing for the validation and verification, it is expected that Poker Playing Agent system will be completed at the end of May 2015. PPA will introduce another perspective to the computer poker literature. System will be built with state-of-the-art algorithms and it will be a chance for other developers and researchers to observe the outcomes in a real world problem. It is planned to share this gained knowledge with the community by publishing a paper at the end of the project. For further information, section “1.Overview” can be examined.

## 4.3 DESIGN STAKEHOLDERS AND THEIR CONCERNS

The stakeholders of the Poker Playing Agent system are forthcoming and current computer poker researchers, poker game developers and advisors and developers of the project. This project is shaped with the research interests of the developers in the guidance of advisors' former experiences and advices. One possible concern of the stakeholders' is the system's reliability. It should be guaranteed that agent will make rational moves, i.e. it will act according to the poker game rules and it will always try to maximize its gains during the game. One another possible concern is the performance of the system; it should give real time responses during the game. Also, stakeholders of the system want to observe the agents gameplay via a graphical tool. They want this PPA system to be completed in time and to be informed during the complete process.

## 4.4 DESIGN VIEWS

Poker Playing Agent system is a research project, which has a continuously evolving nature. Hence during the design and implementation processes, stakeholders may introduce new features or may remove some of existing ones to replace with one another. To facilitate this flexibility, an object oriented and easily extendable modular approach will be adapted while developing the system. To clarify this design, a contextual view to determine and indicate services provided by system and how user interacts with them, a logical view to describe class relations, dependency and patterns use views to define components in and around the system and their interactions, an interface view to visualize the flow of the information during process and an algorithm view to describe the essential algorithms that will be used during the development is needed and will be provided in this SDD.

## 4.5 DESIGN VIEWPOINTS

- *Context viewpoint*: This viewpoint focuses on the services provided by the system, the role of user actor in the system and how actor interacts with it. In this viewpoint, system boundaries, input and output relations between components should also be explained. This viewpoint can be used as a base for the evaluation and analysis of the system.
- *Composition viewpoint*: This viewpoint explains the components of the Poker Playing Agent system and what roles these components have.
- *Logical viewpoint*: Class structure of the system is provided in this viewpoint. Interaction between each class, how they designed and implemented is also specified.
- *Dependency viewpoint*: System environment and the interconnections between entities are specified in this viewpoint.

- *Information viewpoint*: This viewpoint describes how persistent data is stored within the system by providing an Entity Relationship diagram.
- *Pattern use viewpoint*: It addresses design ideas and patterns which make system more transparent to both developers and stakeholders.
- *Interface viewpoint*: This viewpoint specifies the details of the external and internal interfaces. It is also necessary for the proper testing phase of the system. Some mockup visualizations and screenshots will be provided while describing the interfaces.
- *Interaction viewpoint*: It defines the interaction between entities of the system, regarding why, where, how, and at what level actions occur.
- *State dynamics viewpoint*: It is used when internal behavior of the system within its components and with the user is under interest.
- *Algorithm viewpoint*: This viewpoint describes the basic algorithms required to develop the system.

## 4.6 DESIGN ELEMENTS

Design elements occurring in a design view such as design entity, design relationship, design attribute and design constraints will be explained in section "5. Design Viewpoints" in detail. Hence for further information please check that section.

## 4.7 DESIGN RATIONALE

The motivation behind why this project has been chosen and a system has been designed as explained in this SDD document is finding efficient solutions to poker game problem which requires handling unreliable and incomplete information and then creating a step towards the adaptation of these solutions to similar real life problems.

Design choices of this project is based on its research nature and dynamics. A structure supporting Bayesian Poker has been created to allow a rational decision process and effective learning component. However, there will be new requirements as time goes by and new features will be introduced to the system. Hence, design choices are considered to enhance the reusability and to provide extensibility and sustainability. In addition to this, system's overall performance is a base measurement that may affect the general design. After current design being implemented, a review process will be held to consider possible changes in the system and this may affect the structure of the system.

## 4.8 DESIGN LANGUAGES

Unified Markup Language (UML) is selected as a part of design viewpoint specification. This choice is highly related with its well defined semantics and syntax and also its capability to express object-oriented concepts. Throughout the document, many types of the UML diagrams will be used and also Entity Relationship diagrams will be provided when necessary.

# 5 DESIGN VIEWPOINTS

## 5.1 INTRODUCTION

In this section of the SDD, design viewpoints of the project will be explained. Each design viewpoint addresses different perspectives to be focused on to cover system requirements and to discuss these requirements with the stakeholders. Viewpoints that will be covered are listed below:

- Context Viewpoint
- Composition Viewpoint
- Logical Viewpoint
- Dependency Viewpoint
- Information Viewpoint
- Patterns Use Viewpoint
- Interface Viewpoint
- Interaction Viewpoint
- State Dynamics Viewpoint
- Algorithm Viewpoint

During the explanation of viewpoints, UML diagrams will be used in order to increase understandability.

## 5.2 CONTEXT VIEWPOINT

Context viewpoint shows all of the functionalities of the system provided by the design. Also in this section of the SDD, system boundaries and interaction between system and users will be specified.

System environment and how users will interact with it is shown in the diagram below. Details of the system environment can be found in Section 2.1 Product Perspective of the SRS document of this project.

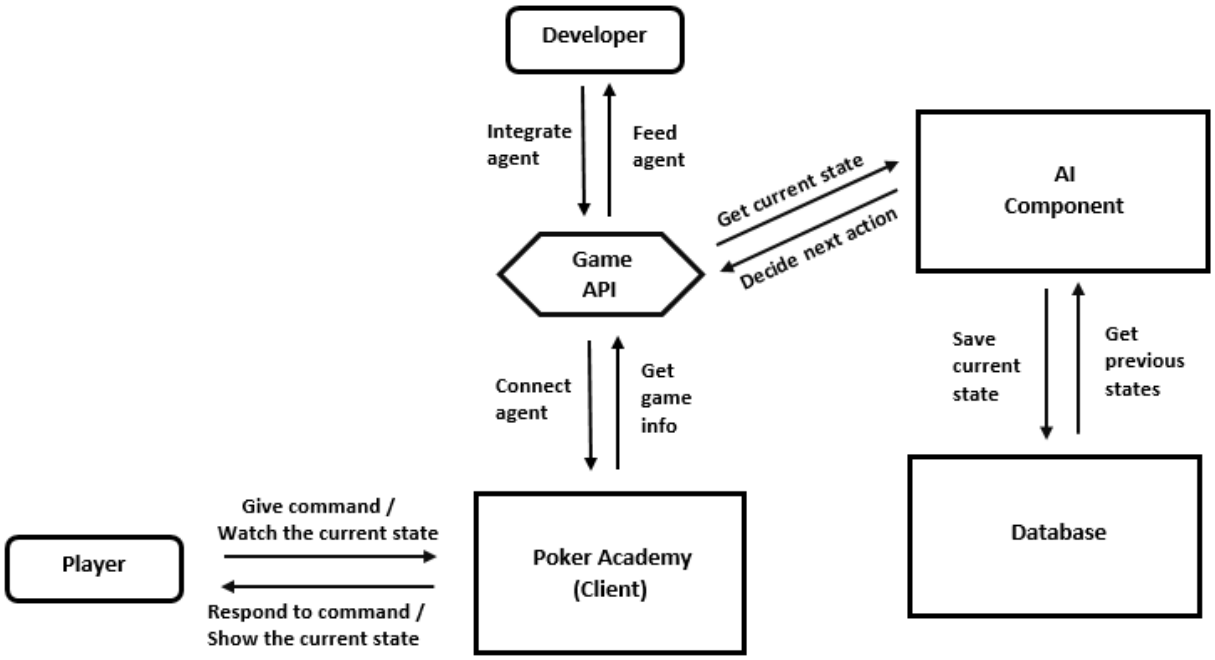


*Figure 1 Context Diagram*

System actors as users can be divided into categories; forthcoming developers and researchers and general poker players. Former category will be able to examine our agent's capabilities and make their own agents play against ours. Latter ones will only be able to play against our and other agents as a human player. There will be some differences in the interaction of the system with different types of users. Developers will interact with the system using Meerkat library. They will be using functionalities adopted from it to create agents and connect these agents to the system. However, players will interact with the system by entering Poker Academy Client and will be treated as regular human players by the system. To conclude, developers will have the abilities of the players with the additions. Use cases can be seen from the diagram below.

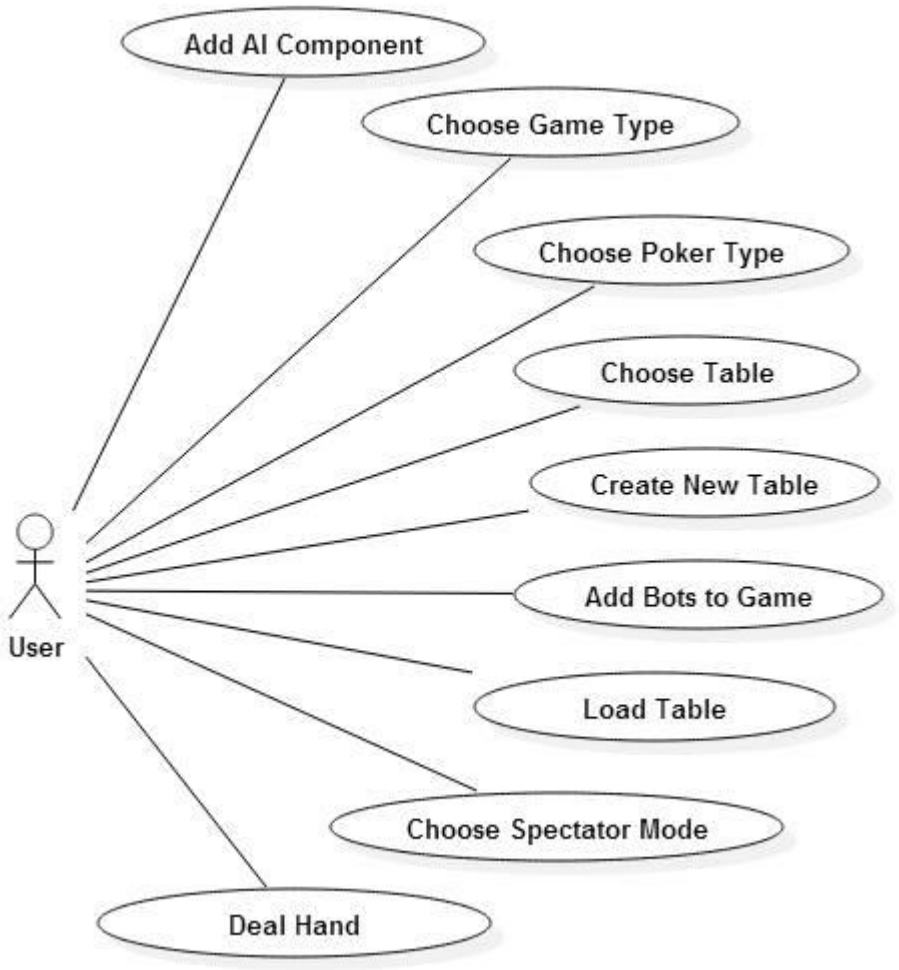


*Figure 2 General Use Case Diagram*

Since use cases are explained in the SRS of the project in detail, they will not be examined here again. For further information, please see Section 2.2 Product Functions of SRS document.

## 5.3 COMPOSITION VIEWPOINT

In this section, the components of the PPA System and their roles are described. A deployment diagram is given below to render these components.

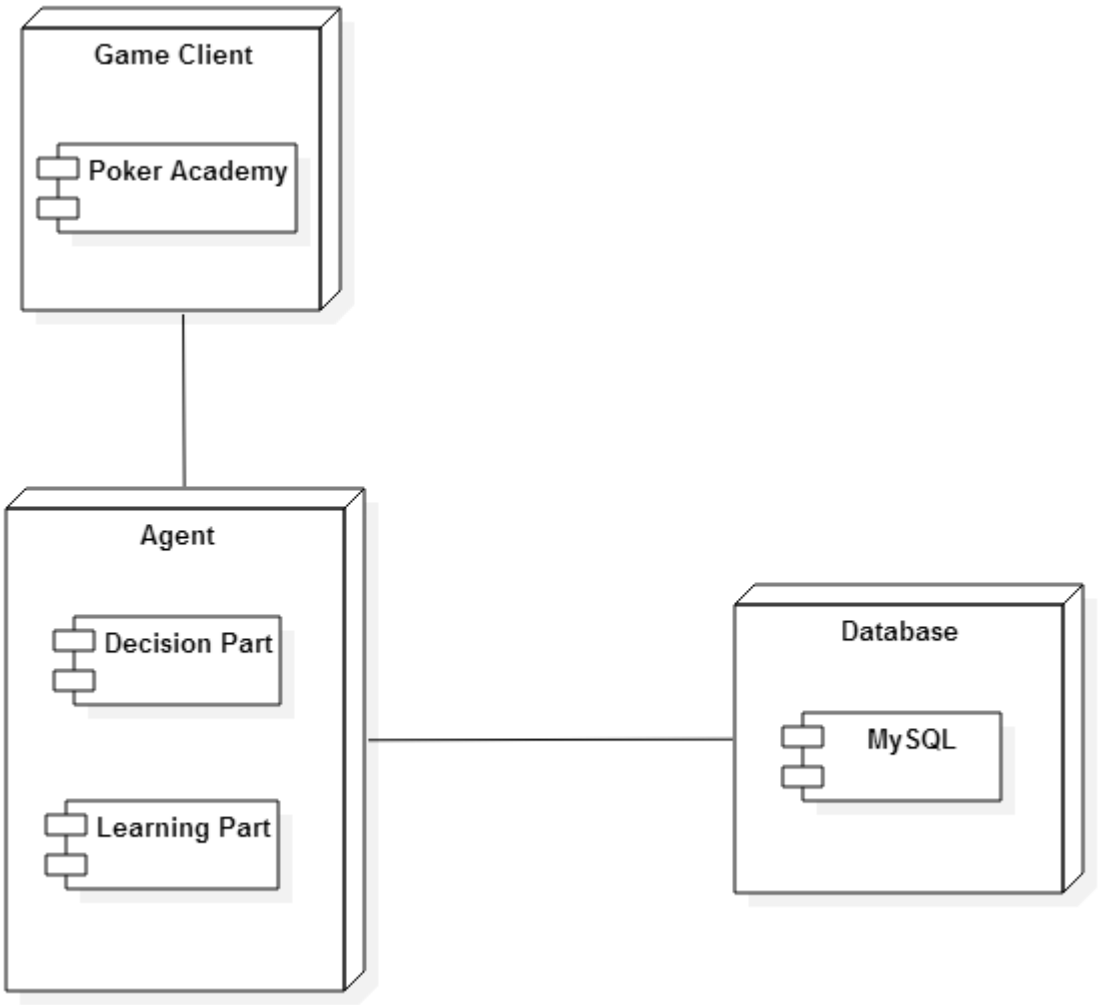


*Figure 3 Deployment Diagram*

### 5.3.1 Game Client

Users will be able to observe and get statistical information about current and past games using Poker Academy. To decide its next action, bot will get necessary information such as community cards, statistical information, current hand information, pot odds via this client application.

### 5.3.2 Agent

#### 5.3.2.1 Decision Part

Main task of this part will be analyze opponent's behaviors, current game state and past game experiences. After that it will choose best next action based on its analyses.

#### 5.3.2.2 Learning Part

Basically, learning part keeps track of the past game information and opponent's behaviors. It stores this information in a Bayesian Network. For each hand learning part has a Bayesian Network.

### 5.3.3 Database

Bayesian Networks which are created by learning part will be kept in the database. When needed necessary information about previous state will be accessed through this database.

Users will interact with the system through Poker Academy as a game client application and the next action for the game is chosen using decision part and learning part of agent and shown with its result to users via Poker Academy. Agent will communicate with database to get necessary information about which action is the most proper for the current state to these parts.

To explain the structure of the system with its classes and interface, a component diagram is provided below.

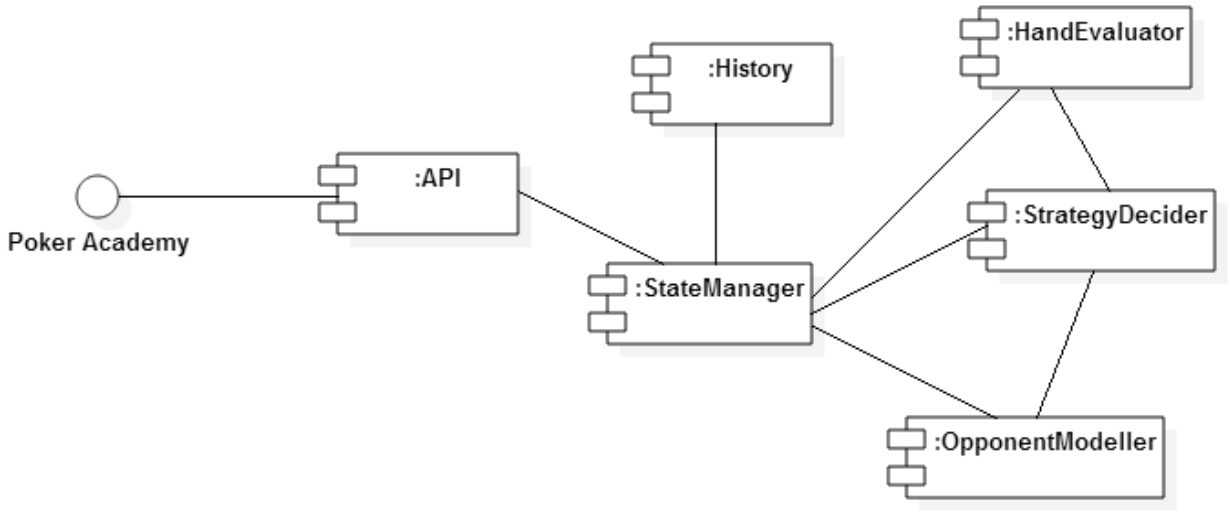


*Figure 4 Component Diagram*

## 5.4 LOGICAL VIEWPOINT

In this section classes of the Poker Playing Agent system will be explained by giving their UML class diagrams. There are five classes in the system. These are

- State Manager Class
- History Class
- Hand Evaluator Class
- Opponent Modeler Class
- Strategy Decider Class
- Meerkat API

After examining each class individually, the relationship between these classes will be provided.

### 5.4.1 Meerkat API

Poker Academy provides a Java based API (named the Meerkat API) that allows poker bot coders to plug in their own custom bots. This gives an excellent environment for bot development as you can easily play against your bot in a quality GUI or have your bot play thousands of hands against the other bots. The program also keeps track of all the hands played and can display comprehensive graphs and analysis of the player statistics.

In this section some major classes will be explained by giving short information and their usage.

#### 5.4.1.1 Action Class

This class manages actions by a player in a poker game. The major features can be listed as:

- Action Filter does some basic checks to change the action if game conditions warrant it.
- Get Action creates an action from classic values.
- Get Action Index converts an update action to a general action {fold,call,raise} or -1 if not a normal voluntary action.

#### 5.4.1.2 Card Class

This class represents a playing card from a set of cards which map to cards having a suit and a face value. The major features can be listed as:

- Get Index return the integer index for this card.
- Get Rank obtains the rank of this card.
- Get Suit obtains the suit of this card.
- Set Card changes this card to another.
- Valid tests if the card is valid.

#### 5.4.1.3 Deck Class

A Deck of 52 Cards which can be dealt and shuffled some functions could be made much faster with some extra memory. The major features can be listed as:

- Deal obtains the next card in the deck.
- Extract Card removes a card from within the deck.
- Find Card finds position of Card in Deck.
- Get Card obtains the card at a specific index in the deck.
- Reset places all cards back into the deck.
- Shuffle places all cards back into the deck and then shuffles the deck.

#### 5.4.1.4 GameInfo Class

This class stores all of the info defining a single game of poker. The major features can be listed as:

- Add Winner adds a player to the list of winners.
- Active Player tests if a player at a specific position is active in the game.
- Big Blind shows big Blind the current player.
- Call calls the current player
- Fold folds the current player
- Get Pot gets the total size of the pot, including all side pots
- Get Board Cards obtains a Hand containing the board cards.

#### 5.4.1.5 Hand Class

This class stores a Hand of Cards (up to a maximum of 7). The major features can be listed as:

- Add Card adds a card to the hand.

- Get Card gets the specified card in the hand.
- Get Card Index get the specified card id.
- Remove Card removes the last card in the hand.
- Make Empty removes the all cards from the hand.
- Size gets the size of the hand.

#### 5.4.1.6 PlayerInfo Class

This class stores all of the information for a player during a poker game. The major features can be listed as:

- Active return true if the player is still active in the hand.
- All In determines if a player is All-In.
- Get Amount in Pot obtains the amount the player has put in the pot..
- Get Amount to Call determines the amount a player must pay to stay in the game.
- Get Game Info gets the context of the last action made by this player.
- Get Last Action returns a single integer code for the last action made.

#### 5.4.1.7 Pot Class

This class handles a Pot data structure, which accumulates money from players, and can break into multiple side pots. The major features can be listed as:

- Remove Uncalled Chips removes all uncalled chips from the pot.

### 5.4.2 State Manager Class

This class manages flow of information between other components and it makes the interaction with Game API.

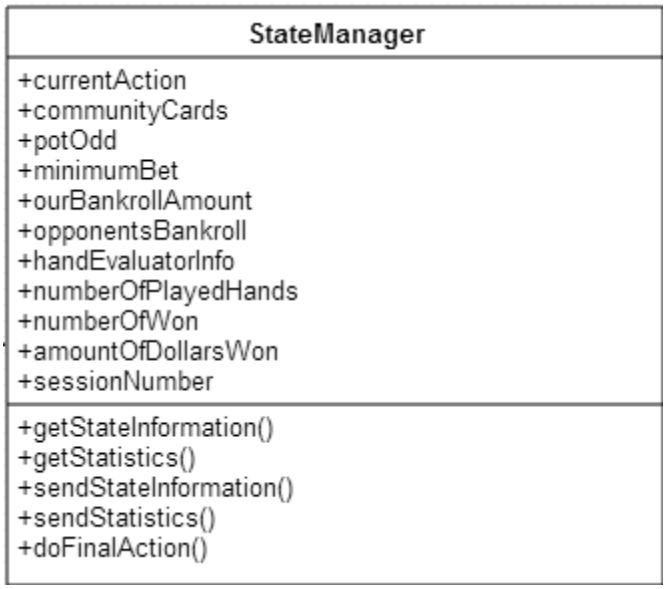


*Figure 5 State Manager Class*

Attributes and methods of State Manager class are examined below.

| **Name** | **Type/Return Value Type** | **Visibility** | **Definition** |
|---|---|---|---|
| currentAction | Struct<String, String, Float, Date> | communityCards | This struct keeps the information about the current action who take(name), what kind of action take(i.e. fold, chech etc.), how much gain(amount of money) with the time staps. |
| communityCards | List<String> | Public | This variable keeps the information about community |

| | | | |
|---|---|---|---|
| | | | cards which can be seen from anyone. |
| potOdd | Float | Public | This variable keeps the information about the total amount of money which is in pot. |
| minimumBet | Float | Public | This variable keeps the information about the minimum amount of money which is required to continue game. |
| ourBankrollAmount | Float | Public | This variable keeps the information about the total amount of money we possess. |
| opponentsBankroll | List<Pair<String, Float>> | public | This variable keeps the information about the total amount of money which each player possess. |
| handEvaluatorInfo | List<Pair<String, Float>> | Public | This variable keeps the information about estimated winning chance which is produced from Game itself. |
| numberOfPlayedHands | Integer | Public | This variable keeps the information the number of game we played. |
| numberOfWon | Integer | Public | This variable keeps the information the |

| | | | |
|---|---|---|---|
| | | | number of game which we won. |
| amountOfDollarsWon | Float | Public | This variable keeps the information about the total amount of money which we gain. |
| sessionNumber | Integer | Public | This variable keeps the information about the number of game we play currently. |
| getStateInformation | Void | Public | This function gets the information from Game API and assign them into its attributes. |
| getStatistics | Void | Public | This function gets statistics which is generated from Game itself through Game API. |
| sendStateInformation | Void | Public | This function sends whole state information to History Class, Strategy Decider Class, Hand Evaluator Class and Opponent Modeler Class. |
| sendStatistics | Void | Public | This function sends whole statistics to History Class, Strategy Decider Class, Hand |

| | | | |
|---|---|---|---|
| | | | Evaluator Class and Opponent Modeler Class. |
| doFinalAction | Void | Public | This function sends the final action which is produced from Strategy decider Class. |

### 5.4.3 History Class

This class keeps the past statistics and state information with time stamps which we use to train our model and also it makes the interaction with State Manager class. It keeps the information inside the database. This class is generalization of State Manager.

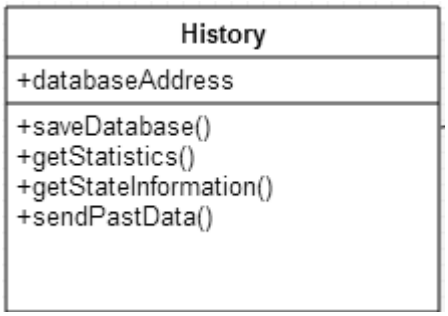


*Figure 6 History Class*

Attributes and methods of History class are examined below.

| **Name** | **Type/Return Value Type** | **Visibility** | **Definition** |
|---|---|---|---|
| databaseAddress | Object | Public | This variable keeps the information about the address of the database |
| saveDatabase | Void | Public | This function saves the information which is taken from State Manager |

| | | | |
|---|---|---|---|
| getStatistics | Void | Public | This function gets the past statistics from database |
| getStateInformation | Void | Public | This function gets the past state information from database |
| sendPastData | Void | Public | This function sends the past data to the state manager. |

### 5.4.4 Hand Evaluator Class

This class uses the current state observations to calculate absolute numerical score. This class is generalization of State Manager.

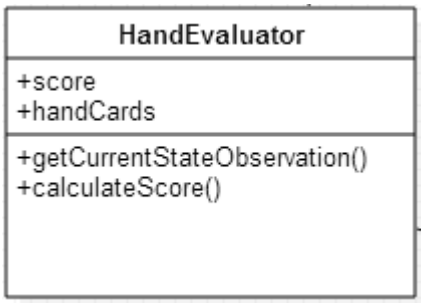


*Figure 7 Hand Evaluator Class*

Attributes and methods of Hand Evaluator class are examined below.

| **Name** | **Type/Return Value Type** | **Visibility** | **Definition** |
|---|---|---|---|
| score | Integer | Public | This variable keeps the absolute numerical score |
| handCards | List<string> | Public | This variable keeps the cards that we have with the community cards |

| | | | |
|---|---|---|---|
| getCurrentStateObservation | List<String> | Public | This function gets the current hand information and set them to the handCards variable |
| calculateScore | Integer | Public | This function gives the hand an absolute numerical score. Higher score = better hand. |

### 5.4.5 Opponent Modeler Class

This class keeps the Bayesian Network. It creates Bayesian Network from the information that we take during the game and past game states information with time-stamps. This network is used for making inference about opponent players. This class is generalization of State Manager.

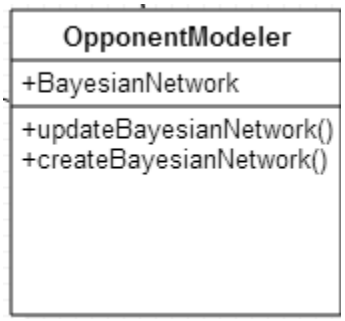


*Figure 8 Opponent Modeler Class*

Attributes and methods of Opponent Modeler class are examined below.

| **Name** | **Type/Return Value Type** | **Visibility** | **Definition** |
|---|---|---|---|
| Bayesian Network | List<List<Integer, Integer, Float>> | Public | This variable keeps probabilistic |

| | | | |
|---|---|---|---|
| | | | graphical model that represents a set of random variables and their conditional dependencies via a directed acyclic graph. The First two integer represents the vertices and the last one represents the weight between these vertices. |
| updateBayesianNetwork | Void | Public | This function updates the Bayesian Network with the given information |
| createBayesianNetwork | Void | Public | This function creates the Bayesian Network. |

### 5.4.6 Strategy Decider Class

This class keeps the probabilistic decision tree. This tree explicitly represent decisions and decision making and the resulting classification tree can be an input for decision making. It creates the tree from the information that are gathered Bayesian Network and past/current state information from State Manager. This class is generalization of State Manager.

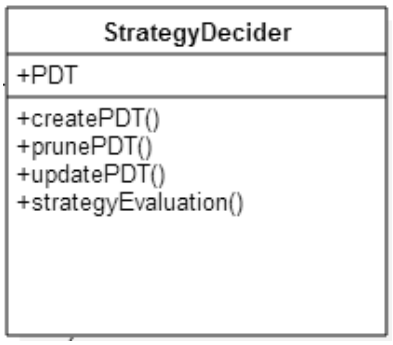


*Figure 9 Strategy Decider Class*

Attributes and methods of Strategy Decider class are examined below.

| **Name** | **Type/Return Value Type** | **Visibility** | **Definition** |
|---|---|---|---|
| PDT | List<List<Integer, Integer, Float>> | Public | This variable keeps a probabilistic decision tree as a predictive model which maps observations about an item to conclusions about the item's target value. The First two integer represents the vertices and the last one represents the weight between these vertices. |
| createPDT | Void | Private | This function creates probabilistic decision tree. |
| updatePDT | Void | Private | This function updates the probabilistic decision tree with the given information |
| prunePDT | Void | Private | With this function we reduces the size of decision trees by removing sections of the tree that provide little power to classify instances. |
| strategyEvaluation | String | Public | This function analyzes the tree and concludes final decision. It returns |

| | | | |
|---|---|---|---|
| | | | Action which is defined in Action enumeration class. |

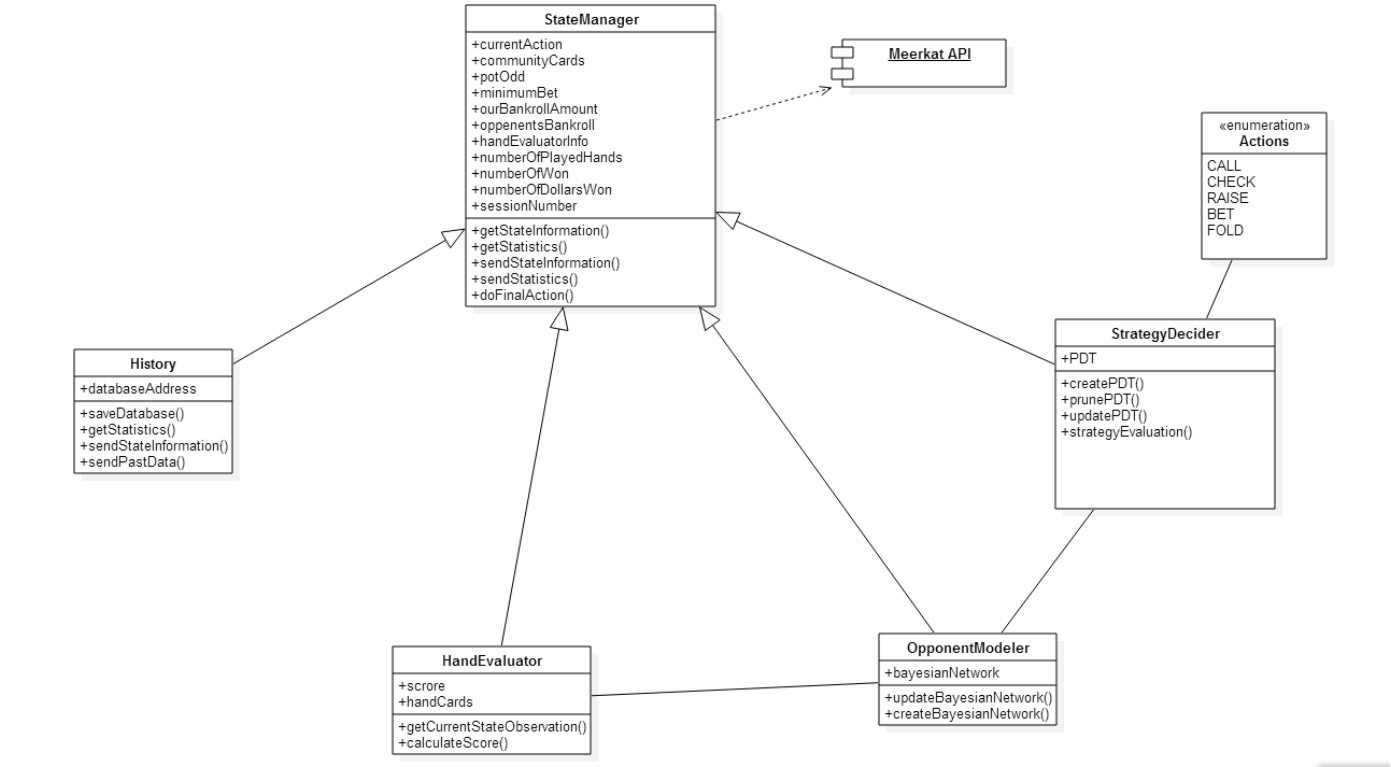


*Figure 10 Class Diagram of the PPA*

## 5.5 DEPENDENCY VIEWPOINT

Dependency viewpoint is related to interconnection and access among entities. Since it is so closed to implementation, this viewpoint will be omitted in this document. Detailed description of entities can be found in "5.4 Logical Viewpoint" section and functionalities and explanation of them in "5.3 Composition Viewpoint". For any further information, please check SRS document and "5.10 Interaction Viewpoint" section of this document.

## 5.6 INFORMATION VIEWPOINT

Poker Playing Agent system has a database to store statistical information of games which will be used in learning component of the agent and will affect the strategy. This database has five tables and detailed explanations of them can be found in the followings. Furthermore, an Entity - Relationship diagram of the database is given in Figure 11.

- History: This table stores the necessary information about in which state which action is selected and next state according to this selection. 'historyID', 'playerStateID', 'action', 'amount' and 'nextPlayerStateID' are the attributes of the table. 'playerStateID' and 'nextPlayerStateID' are the ids for current state and next state respectively. 'action' attribute indicates the type of selected action and 'amount' is used for amount of the chips put into table in case of this selection being bet or raise. 'historyID' is the primary key for this table.
- Player: In this table, 'playerID', 'name', 'description' and 'isAgent' fields are stored. 'name' is the given name for the player and description field can be set especially if player is agent and more specific explanation is needed. 'isAgent' is a boolean value indicating whether player is an agent or a human and 'playerID' is the primary key for this table.
- PlayerState: The information about player state is kept in PlayerState table. 'playerStateID', 'playerID', 'gameStateID', 'handCards', 'turnNo', 'callAmount' and 'bankroll' are the attributes. 'playerID' is the id of the player and 'handCards' are the cards that player have in current state. 'turnNo' indicates which turn current state is. 'callAmount' is the chip amount that keep player in the game for the current state and 'bankroll' is the chip amount that player have. 'playerStateID' is the primary key of this table.
- Game: This table stores the information about the game. 'gameID', 'beginTime' and 'endTime' are the attributes of the table. 'beginTime' and 'endTime' are the timestamps indicating the starting and finishing time of the game. 'gameID' is the primary key of this table.
- GameState: GameState table is used for keeping information about the game state. 'gameStateID', 'communityCards', 'status', 'dealerID', 'sBlindID', 'bBlindID', 'handNo', 'gameID' are the attributes of the table. 'communityCards' attribute is used for the common cards in the table. 'status' indicates whether the game is in preflop, flop, turn or river stage. 'dealerID', 'sBlindID' and 'bBlindID' are the ids of players who are dealer, small blind and big blind in current hand respectively. 'handNo' keeps how manyth current hand is. 'gameID' is the id of the game and 'gameStateID' is the primary key for this table.

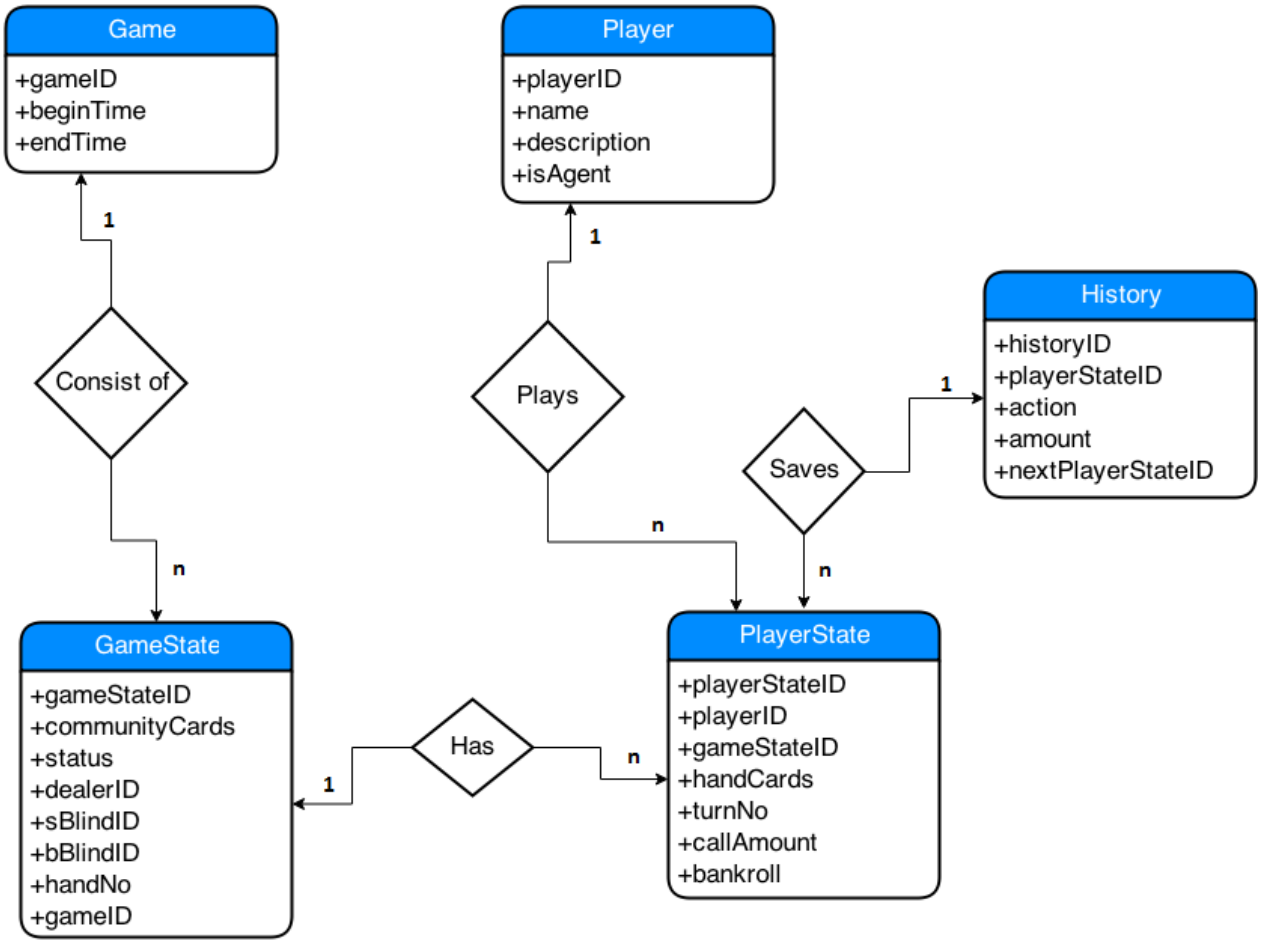


*Figure 11 Entity Relationship Diagram*

## 5.7 PATTERNS USE VIEWPOINT

Design patterns are general reusable solutions to widely generated problems which define how to solve these problems in many different situations. Usage of these patterns make the design more understandable and clear to all stakeholders to discuss on it. In the design of this project, various design and architectural patterns will be used and they will be listed and discussed below.

*Client-Server Pattern:* In Poker Playing Agent system, third party poker game application composes client part and agent component acts as a server. Client, game application, will send request to server for the next applicable action in the game and after this request, agent will compute possibilities and decide next possible effective action. Later, this prepared action response will be sent to client to be executed in the gameplay.

*Strategy Pattern:* In PPA system, agent will be able to choose among many possible poker game actions, such as fold, bet, raise… With the strategy design pattern they will be made them interchangeable by encapsulating them under the same algorithm family. Strategy pattern will let the next action decision vary independently from agent that uses it.

## 5.8 INTERFACE VIEWPOINT

In this section, usage of the services that are provided design is explained by giving the details of the external interfaces of the Poker Playing Agent system. These external interfaces can be classified into two categories. First category includes the interfaces of the Settings part that are responsible for the configuration of the game to be displayed. Second one includes the interfaces of game part that are responsible for displaying the game basically.

In the following sub-sections each interface of Poker Playing Agent system will be examined.

### 5.8.1 Add AI Interface

In this interface, users may develop their own artificial intelligence bots. Using this functionality these different poker bots can be added to system.

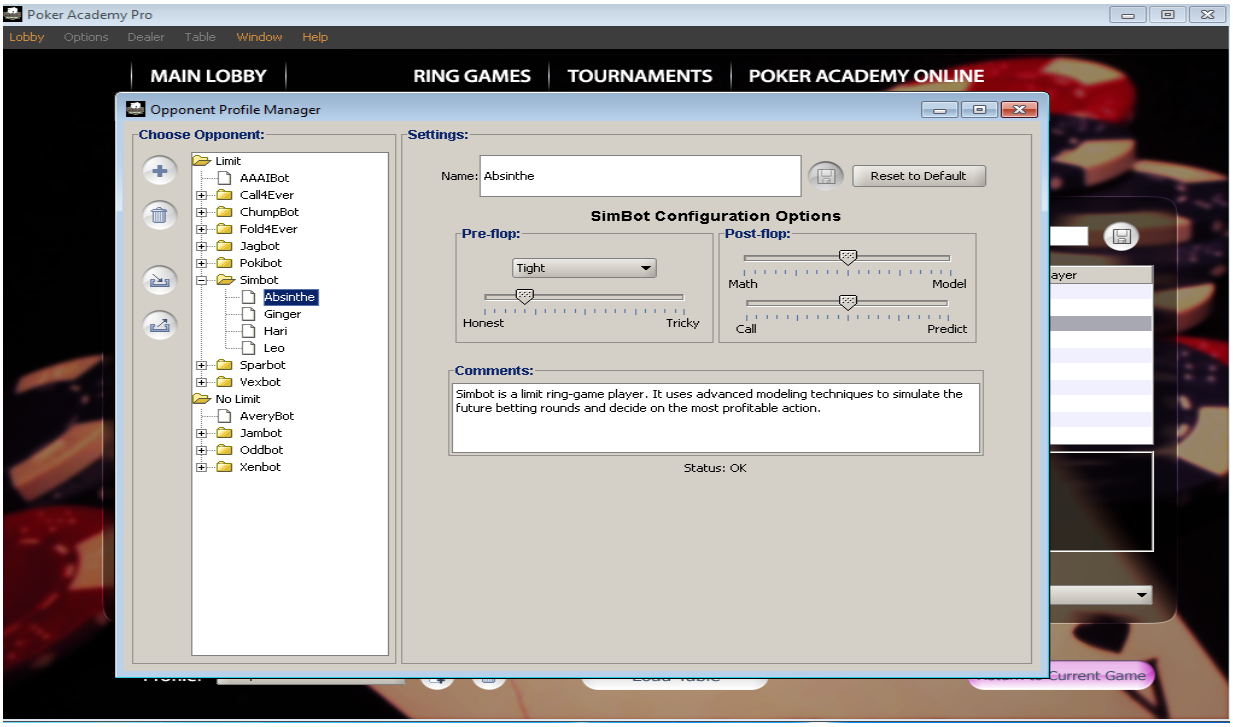


*Figure 12 Add AI Interface*

### 5.8.2 Choose Game Type Interface

In this interface, user can choose type of the poker game by using this functionality. User can choose Ring Games, Tournaments, Poker Academy Online.

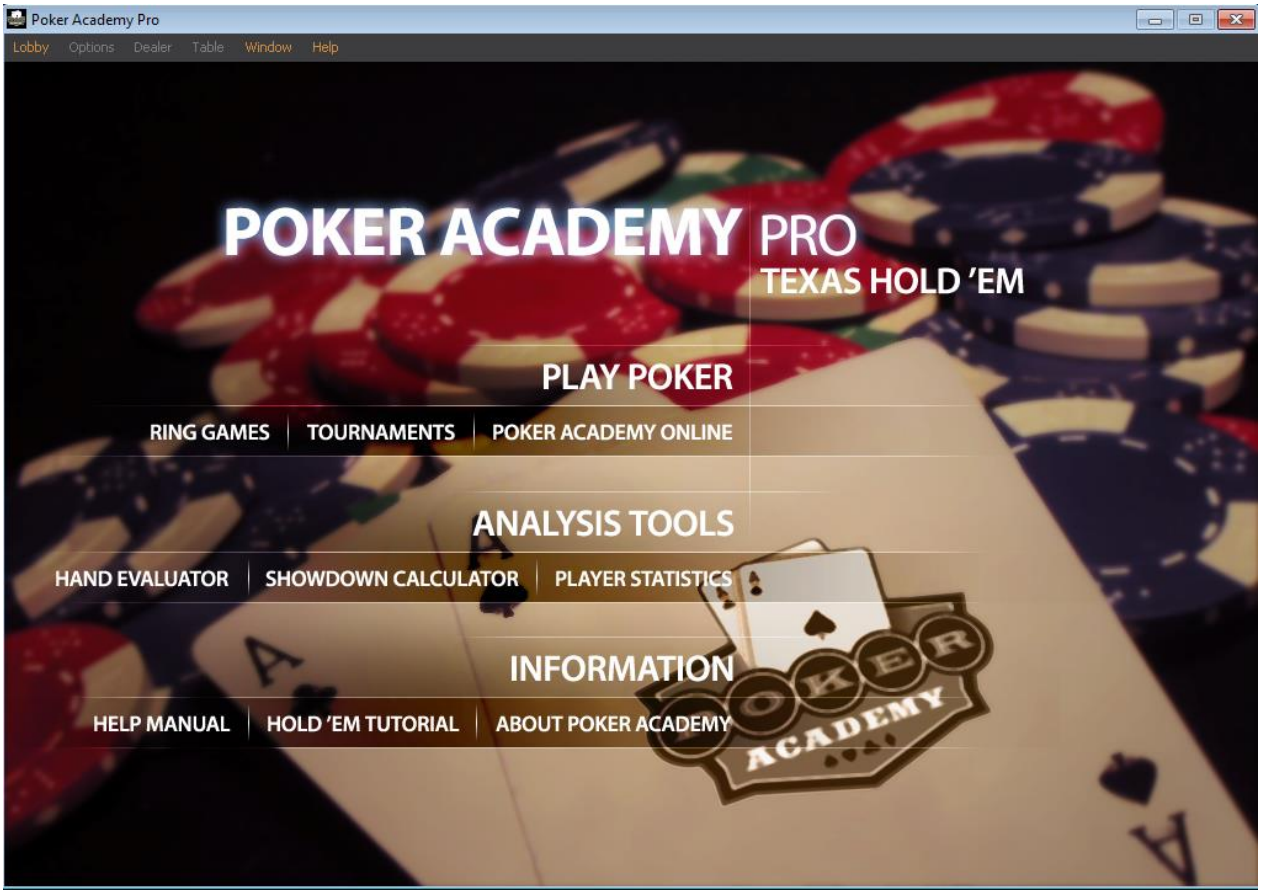


*Figure 13 Choose Game Type Interface*

### 5.8.3 Choose Poker Type Interface

In this interface, after choosing "Ring Games" option in the main menu, user will be able to choose which type of poker he/she and bots will be play. There are options which are "Limit" and "No-Limit" are shown to the user.

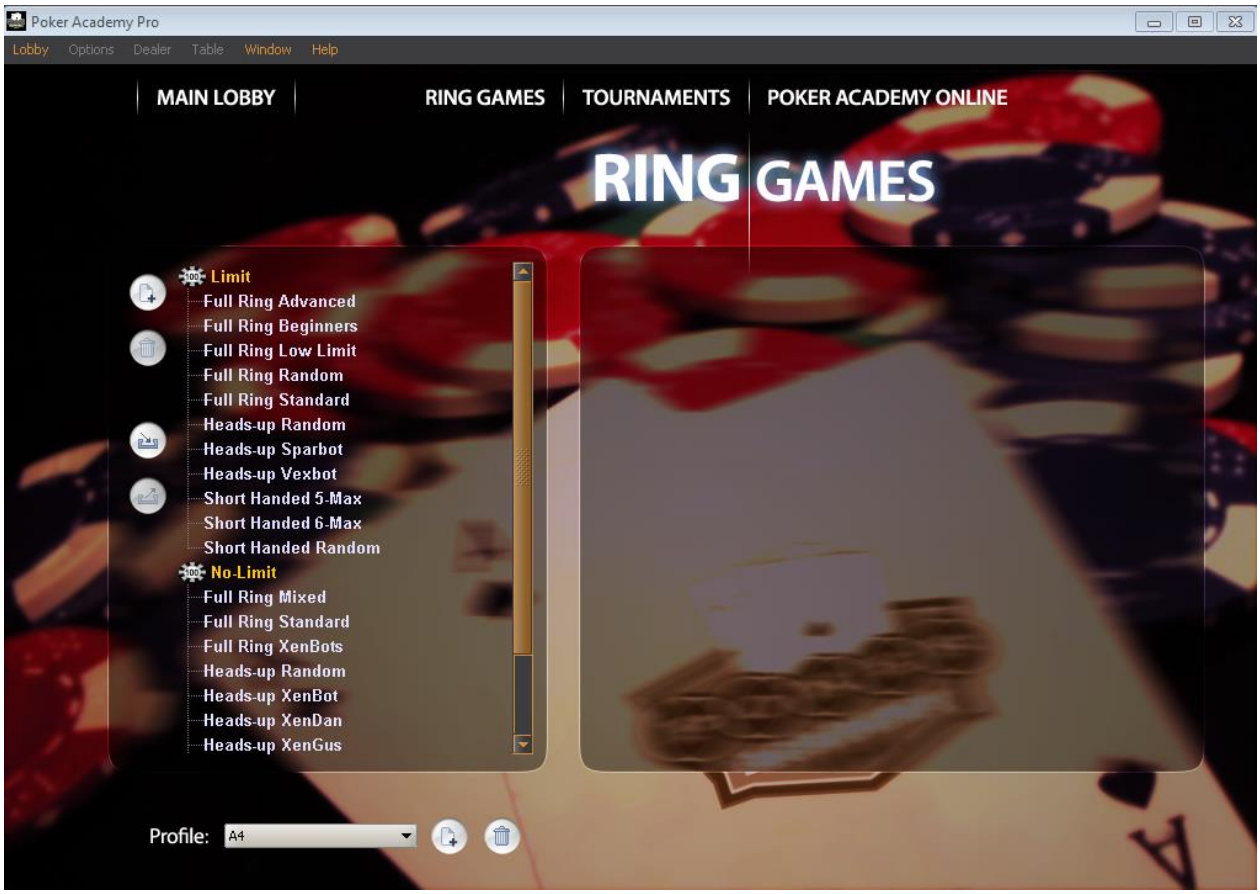


*Figure 14 Choose Poker Type Interface*

### 5.8.4 RING Games Interface

In this interface, under the “Limit” and “No-Limit” options, user will be able to choose an existing game table. There is a set a bankroll button to change bankroll amount. User may change these initial configurations of the tables and may join with his/her own bot to the game. And also, user can create new table by clicking on create table button which is on top left of the page.

There is a load game button, user can change existing or new created table`s configuration before loading the game.

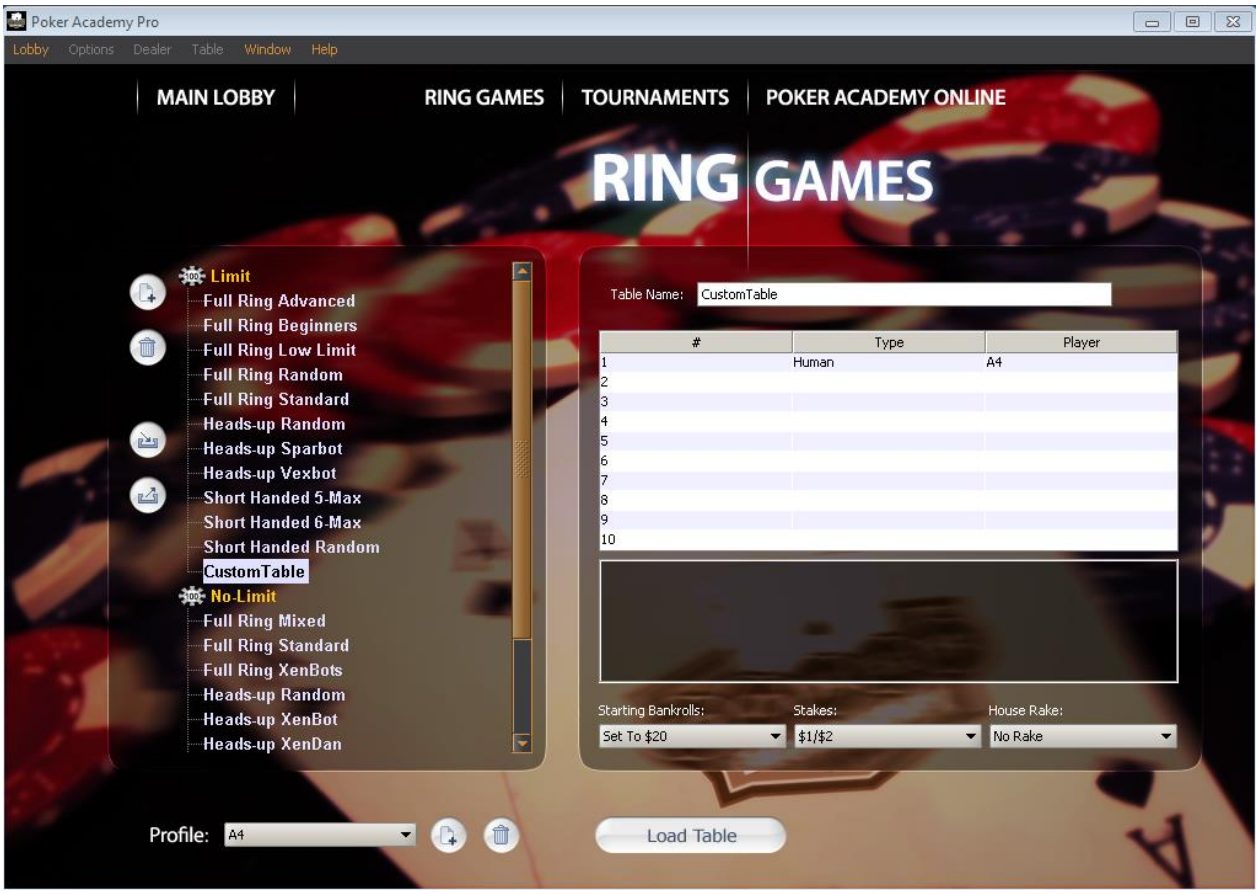


*Figure 15 RING Games Interface*

### 5.8.5 Game Table Interface

In this interface,

- There is a table in the center of the page which game will held.
- There are chairs around table which for player.
- There is a deal hand button to start the game.
- There is ribbon button on the right side of the game by this button user can exit game, see statistics, use calculator and inspects some graphics which is related with current game.
- Again on the right panel there is a real time advisor. User can read scripts from this panel.


*Figure 16 Game Table Interface*

## 5.9 INTERACTION VIEWPOINT

Interaction viewpoint is provided through sequence diagrams to explain the main functionalities of the Poker Playing Agents project. Details of the each viewpoint can be found in the "2.2 Product Functions" section of SRS.

### 5.9.1 Add AI Component

This section refers to 2.2.1 section of the SRS document. In order add an AI bot:

- User puts the configuration file of the bot, namely pd file, into the “bots” folder which is located in “PokerAcademyPro2/data/”.
- User clicks “Opponent Manager” button in the game menu.
- User clicks the “Import” button and choose pd file of the bot.
- User clicks “Save” button.

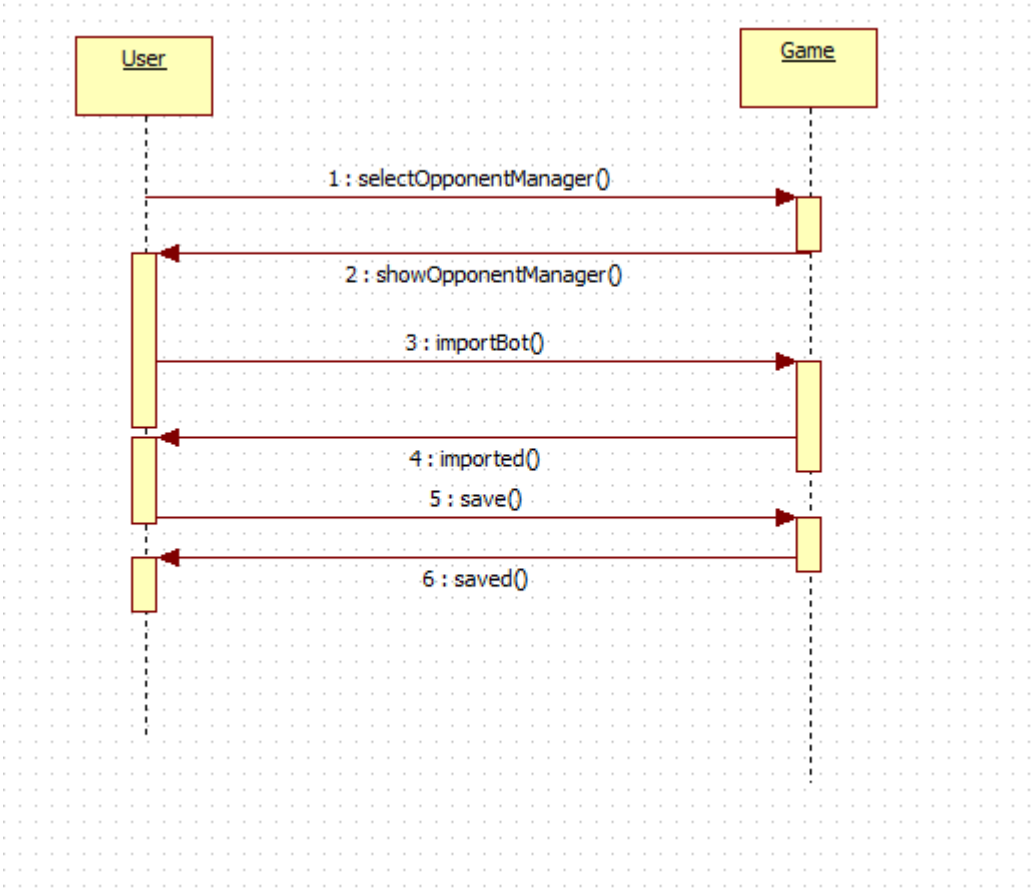


*Figure 17 Interaction 1*

### 5.9.2 Choose Game Type

This section refers to 2.2.2 section of the SRS document. User can choose type of the poker game by using this functionality. After the poker academy is started three different game types at the main menu which are “Ring Games”, “Tournament”, “Poker Academy Online” are shown to the user. To adapt and play their own artificial intelligence bots:

- User should be entered in “RingGames”. User can choose this mode by clicking the button “Ring Games”.

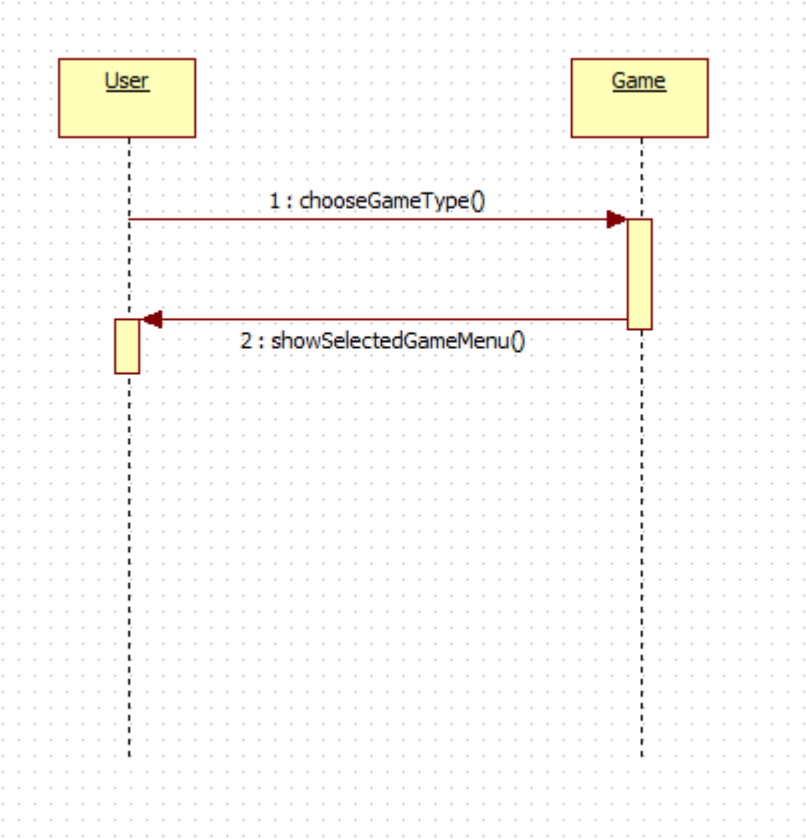


*Figure 18 Interaction 2*

### 5.9.3 Choose Poker Type

This section refers to 2.2.3 section of the SRS document. Since there is no specific action to choose poker type, sequence diagram of this use case does not provided. Choose poker type action can be performed by choose table action.

### 5.9.4 Choose Table

This section refers to 2.2.4 section of the SRS document.
Under the “Limit” and “No-Limit” options, user will be able to choose an existing game table:

- User clicks the name of the table which are created according available bots capabilities, initial bankroll value and number of players.

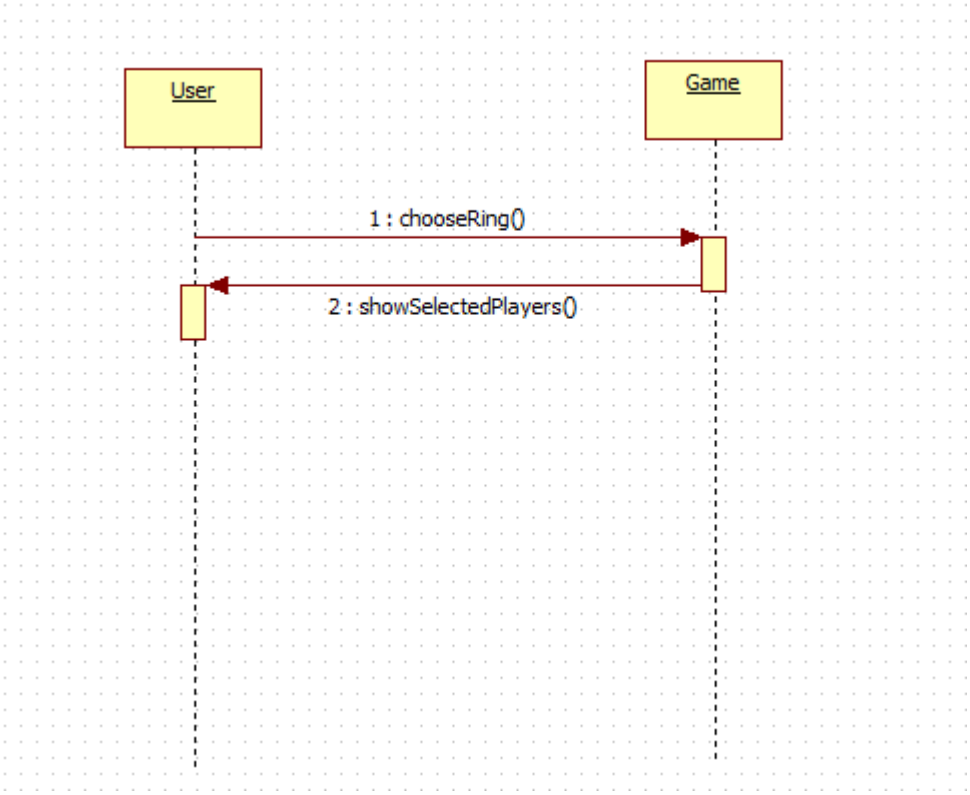


*Figure 19 Interaction 3*

### 5.9.5 Create New Table

This section refers to 2.2.5 section of the SRS document. This functionality is for the user who wants to create a new table instead of choosing an existing game table. In order to create a table:

- User clicks “Create a new table” button.
- User can add bots to the game table, set initial bankroll, stakes and house rake amounts by clicking the dropdown menu of these fields.

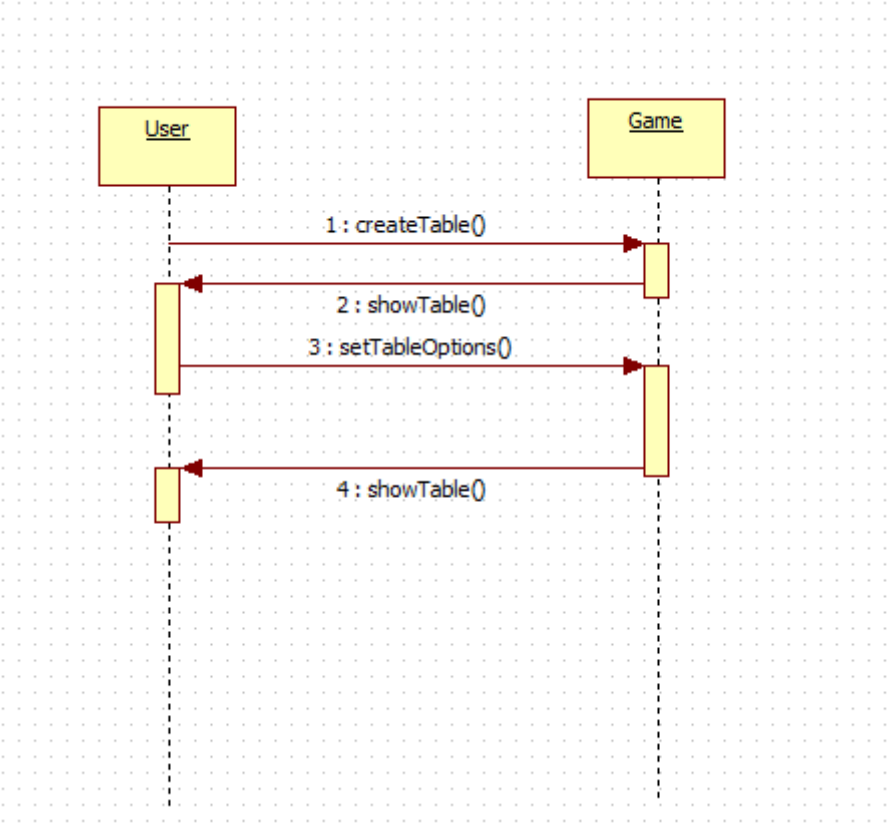


*Figure 20 Interaction 4*

### 5.9.6 Add Bots to Game Table

This section refers to 2.2.6 section of the SRS document. By using this functionality user can add bots he/she chooses to the table. In order to do this:

- User clicks right button on the player list layout.
- A list of available AI bots on the system is shown to the user.
- User can choose their own bots or another famous bots in the literature to add game configuration.

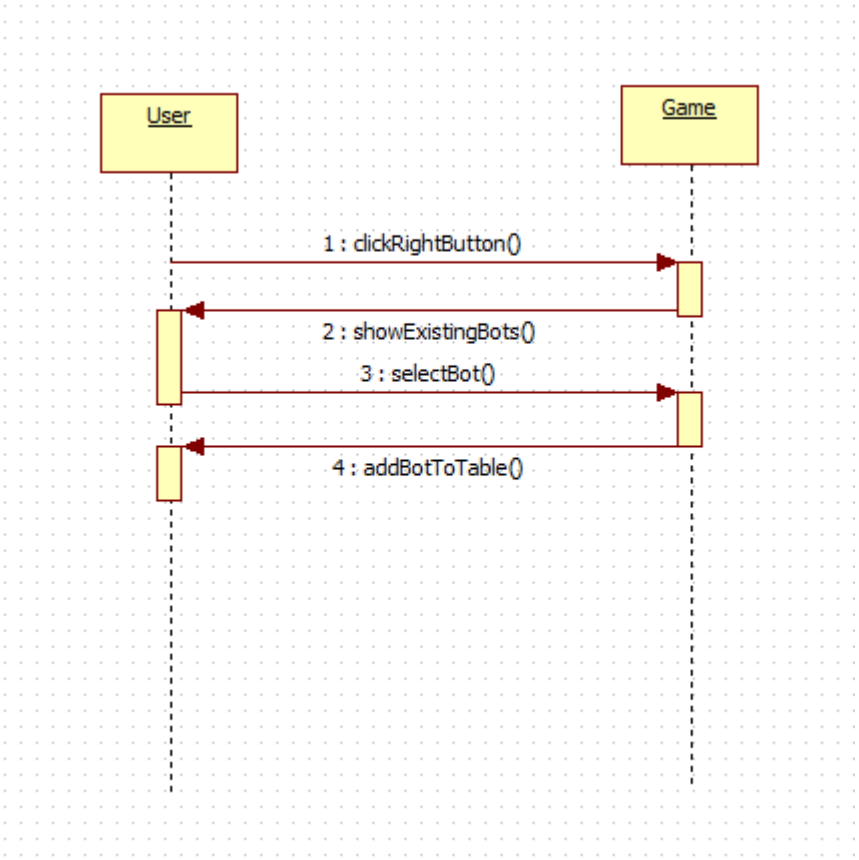


*Figure 21 Interaction 5*

### 5.9.7 Load Table

This section refers to 2.2.7 section of the SRS document. After a table chosen or the configuration of a new table is done in order to start the game:

- User clicks “Load Table” button or double clicks the name of the table. Initial game screen will be created according to chosen bots and bankroll amount and game will be ready to start.

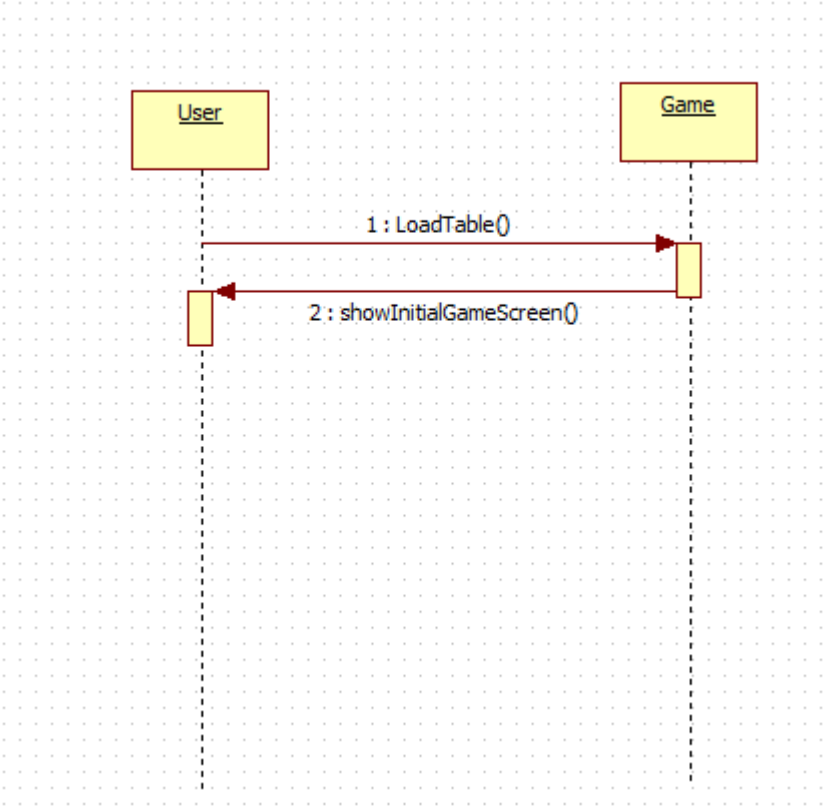


*Figure 22 Interaction 6*

### 5.9.8 Choose Spectator Mode

This section refers to 2.2.8 section of the SRS document. To become an observer during the game:

- User clicks his/her player name and set bankroll from appearing menu.
- User enters “0” as bankroll amount and clicks “OK” button.
- After that if user does not want the game flow to be interrupted, he/she clicks “Options” Menu and then “Auto Deal”.

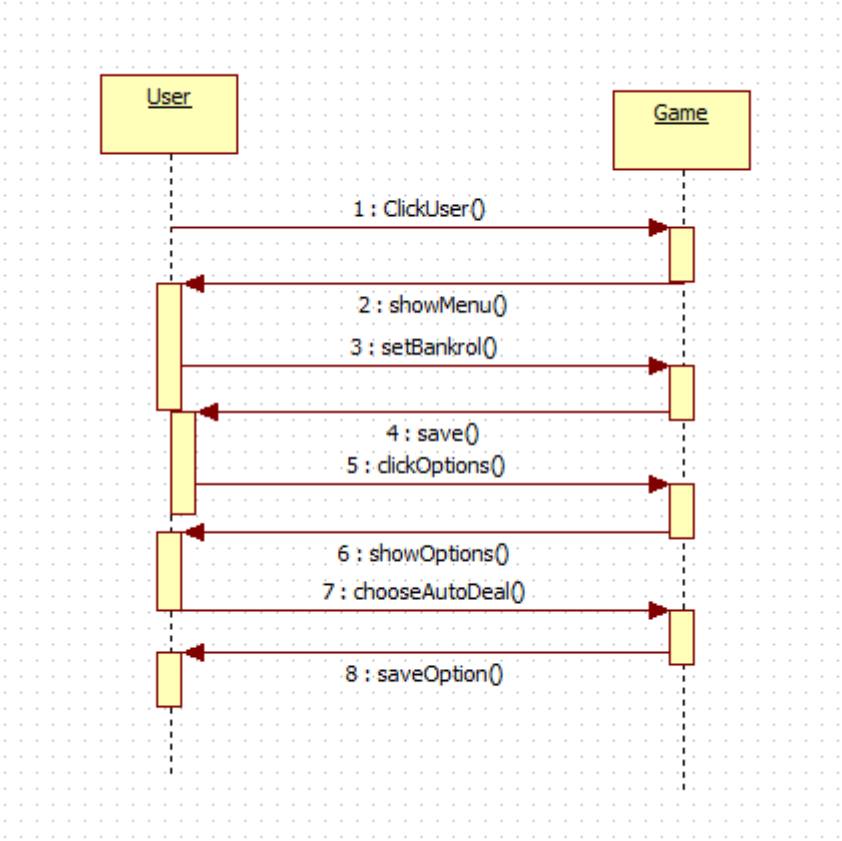


*Figure 23 Interaction 7*

### 5.9.9 Deal Hand

This section refers to 2.2.9 section of the SRS document. After game table is available for user, he/she may manage card dealing process:

- If “Auto Deal” option is selected in the game menu, it must be clicked as mentioned Previous use case again so that it is closed. Closing these option will enable user to use Analytical tools game will provide, such as “Player Statistics”, “Hand Evaluator” for each Hand separately.
- After the auto deal is closed user clicks “Deal Hand” button to start a hand in the game.
- Even if user wants to use auto deal property user must click “Deal Hand” button to start first hand of the game.

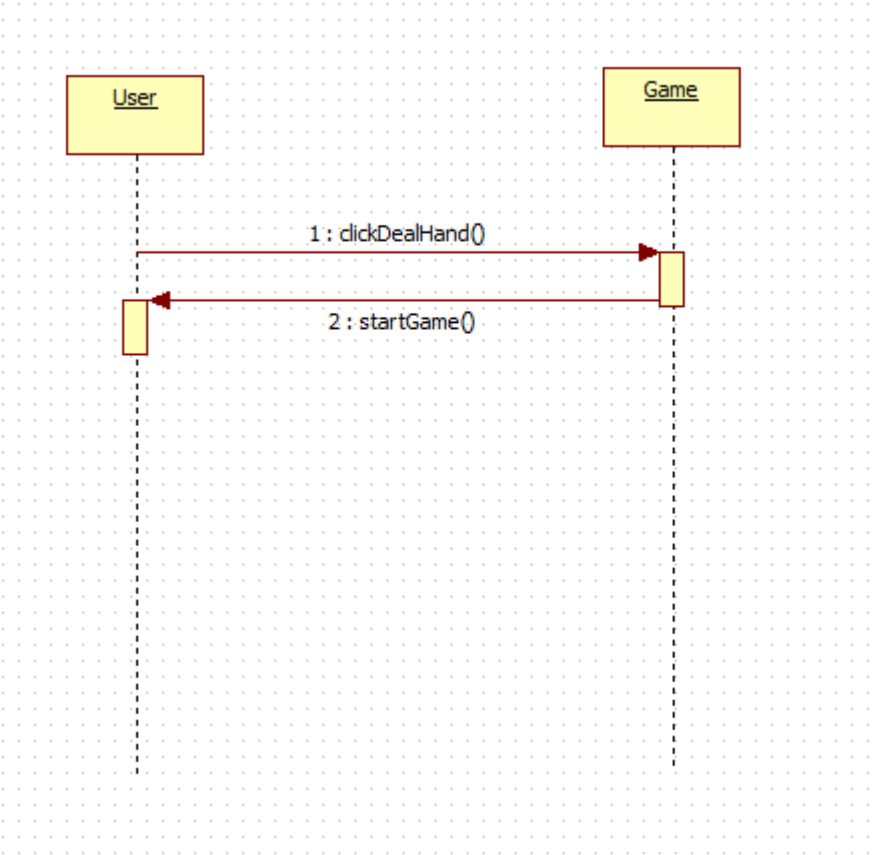


*Figure 24 Interaction 8*

## 5.10 STATE DYNAMICS VIEWPOINT

This viewpoint indicates the dynamic behavior of the system as a response to specific events which can be internal or result of the interaction with the user. States and transitions between them which occur under the effects of these specific events are shown in the following state diagram.

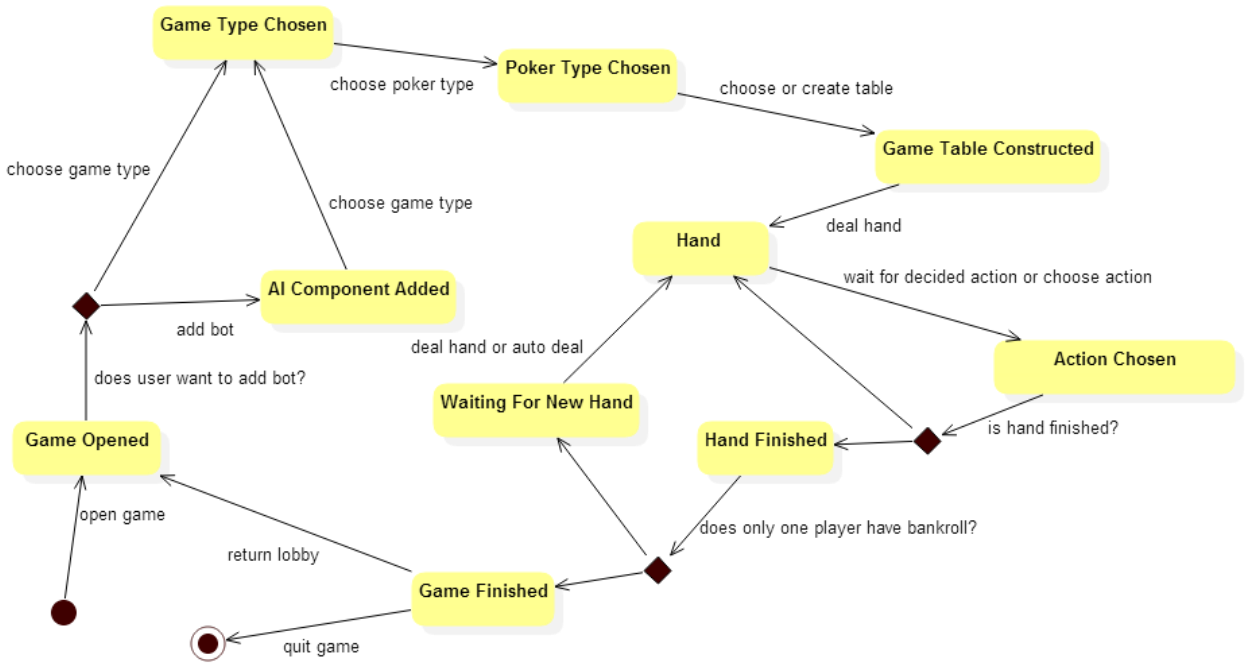


*Figure 25 State Diagram*

## 5.11ALGORITHM VIEWPOINT

The algorithm viewpoint provides details needed by agent developers and analyst of the agent in regard to time-space and logical decision performance. In this viewpoint, general artificial intelligence algorithms and how they will be used in Poker Playing Agent system will be discussed.

*Partially Observable Markov Decision Process:* Markov Decision Process provides a mathematical framework for modeling decision-making in situations where outcomes are partly random and partly under the control of the decision maker. MDP's are useful for studying a wide range of real world problems solved via dynamic programming and reinforcement learning. More precisely a Markov Decision Process is a discrete time probabilistic decision process characterized by a set of states. In each state, there are several actions from which the decision maker must choose. For a state s and an action a, a state transition function determines the transition probabilities to the next state. After the next state is entered the decision maker earns a reward which depends on the new state. The states of an MDP possess the Markov property. This means that if the current state of the MDP at time t is known, transition probabilities to a new state at time t + 1 are independent of all previous states. A Partially Observable Markov Decision Process (POMDP) is

an extension of a Markov Decision Process including hidden information about states. POMDPs are used for choosing actions when the entire world, or state space, is not always directly observable. In other words, you cannot always immediately know where you are in the world and what is going on. Poker is partially observable: you can narrow down which cards are in your opponents' hands from your cards, their bids, and the cards on the table. However, you cannot directly observe which cards are still in their hands. In Poker Playing Agent system, this methodology will be used in the decision component of the agent.

*Bayesian Networks:* Bayesian networks (BNs) belong to the family of probabilistic graphical models. These graphical structures are used to represent knowledge about an uncertain domain. In particular, each node in the graph represents a random variable, while the edges between the nodes represent probabilistic dependencies among the corresponding random variables. These conditional dependencies in the graph are often estimated by using known statistical and computational methods. Hence, BNs combine principles from graph theory, probability theory, computer science, and statistics. A BN reflects a simple conditional independence statement. Namely that each variable is independent of its non-descendants in the graph given the state of its parents. This property is used to reduce, sometimes significantly, the number of parameters that are required to characterize the joint probability distributions of the variables. This reduction provides an efficient way to compute the posterior probabilities given the evidence. In computer poker, this reduction provides an efficiency in the data storage and enables agents to be fed with less complex information. This enables the creation of efficient agents in terms of real time responses and effectiveness since noisy and irrelevant data is eliminated when building BN. In Poker Playing Agent system, this methodology will be used in both decision and learning component of the agent.

Here is the diagram that explain how Bayesian Network will be constructed and the explanation of the nodes in the network can be found below of it.

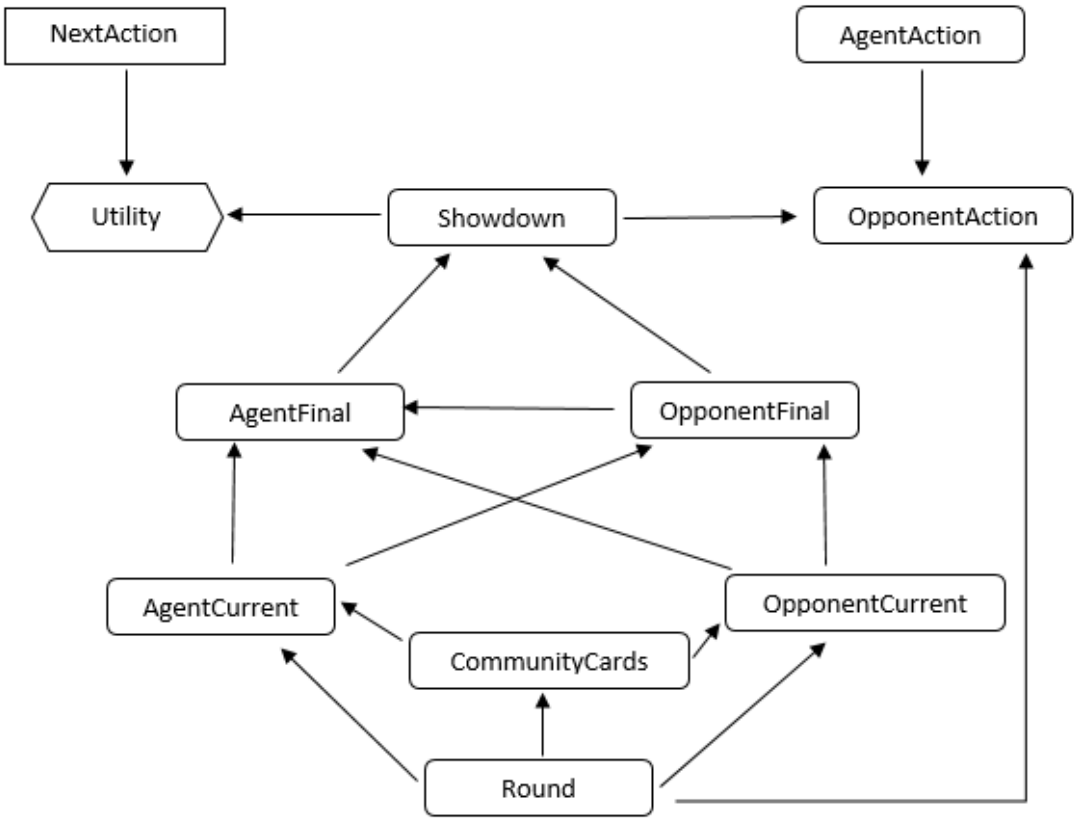


*Figure 26 Structure of BN*

*NextAction:* Valid actions which are made during betting rounds are stored in this node.

*Utility:* It selects best action to take by analyzing incoming state probabilities of the Win node, the available actions from NextAction and the estimated winnings to output a series of utility values for each action.

*Showdown:* State of this node is decided who has the better hand.

*AgentFinal:* Represents agent's final hand.

*OpponentFinal:* Represents opponent's final hand.

*AgentAction:* It represents last action of the agent.

*OpponentAction:* It represents last action of the opponent.

*AgentCurrent:* Represent partial hands for the agent. It is affected by the CommunityCards and Round nodes.

*OpponentCurrent:* Represent partial hands for the opponent. It is affected by the CommunityCards and Round nodes.

*CommunityCards:* It represents current cards on the board.

*Round:* Current round.

# 6 TRACEBILITY MATRIX

| | UC. 1 In SRS | UC. 2 In SRS | UC. 3 In SRS | UC. 4 In SRS | UC. 5 In SRS | UC. 6 In SRS | UC. 7 In SRS | UC. 8 In SRS | UC. 9 In SRS |
|---|---|---|---|---|---|---|---|---|---|
| Figure 1 | | X | X | | | X | | X | |
| Figure 2 | X | X | X | X | X | X | X | X | X |
| Figure 3 | X | X | X | X | X | X | X | X | X |
| Figure 4 | X | X | X | X | X | X | X | X | X |
| Figure 5 | | | | | | | | X | X |
| Figure 6 | X | X | X | X | X | X | X | X | X |
| Figure 7 | | | | | X | X | X | X | X |
| Figure 8 | | | | | X | X | X | | X |
| Figure 9 | | | | | X | X | X | | X |
| Figure 10 | | X | X | | | | X | | |
| Figure 11 | X | X | X | X | X | X | X | X | X |
| Figure 12 | | | | | | | | X | X |
| Figure 13 | | | X | | | X | X | X | |
| Figure 14 | X | X | X | X | X | X | X | X | X |
| Figure 15 | X | X | X | X | X | X | X | X | X |

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| Figure 16 | X | | | | | | | | |
| Figure 17 | | X | | | | | | | |
| Figure 18 | | | X | | | | | | |
| Figure 19 | | | | X | | | | | |
| Figure 20 | | | | | X | | | | |
| Figure 21 | | | | | | X | | | |
| Figure 22 | | | | | | | X | | |
| Figure 23 | | | | | | | | X | |
| Figure 24 | | | | | | | | | X |
| Figure 25 | X | X | X | X | X | X | X | X | X |