

# Design Document

November 7, 2015

SE 2XA3

Hui Chen

Nareshkumar Maheshkumar

Sam Hamel

chenh43

mareshn

hamels2

## Contents

<b>1</b>	<b>Revision History</b>	<b>4</b>
<b>2</b>	<b>Module Guide For No Limit Texas HoldEm Program(NLTP)</b>	<b>5</b>
2.1	Introduction . . . . .	5
2.2	Anticipated and Unlikely Changes . . . . .	5
2.2.1	Anticipated Changes . . . . .	5
2.2.2	Unlikely Changes . . . . .	5
<b>3</b>	<b>Module Hierarchy</b>	<b>6</b>
<b>4</b>	<b>Connection Between Requirements and Design</b>	<b>6</b>
<b>5</b>	<b>Module Decomposition</b>	<b>7</b>
5.1	Hardware Hiding Module . . . . .	7
5.1.1	Texas Holdem Module . . . . .	7
5.1.2	Pot Module . . . . .	7
5.1.3	Game module . . . . .	7
5.2	Behaviour-Hiding Module . . . . .	7
5.2.1	Hand Module . . . . .	8
5.2.2	Deck Module . . . . .	8
5.2.3	Player Module . . . . .	8
5.2.4	AIOpponent Module . . . . .	8
5.2.5	Card Module . . . . .	9
5.3	Software Decision Module . . . . .	9
<b>6</b>	<b>MIS of TexasHoldEm Module</b>	<b>9</b>
6.1	Exported Access Programs . . . . .	9
6.2	Interface Semantics . . . . .	9
6.2.1	State Variables . . . . .	9
6.2.2	Environment Variables . . . . .	9
6.2.3	Assumption . . . . .	9
6.2.4	Access Program Semantics . . . . .	10
<b>7</b>	<b>MIS of Pot Module</b>	<b>10</b>
7.1	Exported Access Programs . . . . .	10
7.2	Interface Semantics . . . . .	10
7.2.1	State Variables . . . . .	10
7.2.2	Environment Variables . . . . .	11
7.2.3	Assumption . . . . .	11

7.2.4	Access Program Semantics . . . . .	11
<b>8</b>	<b>MIS of Game Module</b>	<b>11</b>
8.1	Exported Access Programs . . . . .	11
8.2	Interface Semantics . . . . .	12
8.2.1	State Variables . . . . .	12
8.2.2	Environment Variables . . . . .	12
8.2.3	Assumption . . . . .	12
8.2.4	Access Program Semantics . . . . .	12
<b>9</b>	<b>MIS of Deck Module</b>	<b>13</b>
9.1	Exported Access Programs . . . . .	13
9.2	Interface Semantics . . . . .	13
9.2.1	State Variables . . . . .	13
9.2.2	Environment Variables . . . . .	13
9.2.3	Assumption . . . . .	13
9.2.4	Access Program Semantics . . . . .	13
<b>10</b>	<b>MIS of Card Module</b>	<b>13</b>
10.1	Exported Access Programs . . . . .	13
10.2	Interface Semantics . . . . .	13
10.2.1	State Variables . . . . .	13
10.2.2	Environment Variables . . . . .	14
10.2.3	Assumption . . . . .	14
10.2.4	Access Program Semantics . . . . .	14
<b>11</b>	<b>MIS of Player Module</b>	<b>14</b>
11.1	Exported Access Programs . . . . .	14
11.2	Interface Semantics . . . . .	14
11.2.1	State Variables . . . . .	14
11.2.2	Environment Variables . . . . .	15
11.2.3	Assumption . . . . .	15
11.2.4	Access Program Semantics . . . . .	15
<b>12</b>	<b>MIS of AIOpponent Module</b>	<b>15</b>
12.1	Exported Access Programs . . . . .	15
12.2	Interface Semantics . . . . .	15
12.2.1	State Variables . . . . .	15
12.2.2	Environment Variables . . . . .	15
12.2.3	Assumption . . . . .	15

12.2.4	Access Program Semantics . . . . .	16
<b>13</b>	<b>MIS of Hand Module</b>	<b>16</b>
13.1	Exported Access Programs . . . . .	16
13.2	Interface Semantics . . . . .	16
13.2.1	State Variables . . . . .	16
13.2.2	Environment Variables . . . . .	16
13.2.3	Assumption . . . . .	16
13.2.4	Access Program Semantics . . . . .	16
<b>14</b>	<b>Use Hierarchy</b>	<b>17</b>
<b>15</b>	<b>Traceability Matrix</b>	<b>17</b>
<b>16</b>	<b>Schedule</b>	<b>18</b>

## List of Figures

1	Use Hierarchy Diagram . . . . .	17
---	---------------------------------	----

## List of Tables

1	Revision History . . . . .	4
2	Module Hierarchy . . . . .	6
3	Exported Access Programs for TexasHoldEm Module . . . . .	9
4	Exported Access Programs for Pot Module . . . . .	10
5	Exported Access Programs for Game Module . . . . .	11
6	Exported Access Programs for Deck Module . . . . .	13
7	Exported Access Programs for Card Module . . . . .	14
8	Exported Access Programs for Player Module . . . . .	14
9	Exported Access Programs for AIOpponent Module . . . . .	15
10	Exported Access Programs for Hand Module . . . . .	16
11	Traceability Matrix Between the Module And Requirements .	18
12	Traceability Matrix Between Anticipated Changes And Re- quirements . . . . .	18

## 1 Revision History

Table 1: Revision History

Revision	Date	Team Member	Description of Change
0	November 2	Hui Chen	Started Design Document
0	November 4	Sam Hamel	Design Document Revision 0 Finished
0	November 4	Nareshkumar Maheshkumar	Edit and Proofread Revision 0
1	November 6	Hui Chen	Update MIS Section
1	November 6	Nareshkumar Maheshkumar	Design Document Revision 1 Finished
1	November 6	Sam Hamel	Edit and Proofread Revision 1

## **2 Module Guide For No Limit Texas HoldEm Program(NLTP)**

### **2.1 Introduction**

No Limit Texas HoldEm program follows model decomposition based on the principle of information hiding laid out by Parnas. The system is designed to be adaptable because possible future changes have been accounted for in the form of secrets. This flexibility will allow for easy revision control and will allow developers to quickly modify the program as needed. This design document includes in it a module guide (MG) of the system which gives a brief overview of the modules and allows both designers and maintainers to easily identify the parts of the software. The MG directly relates to the SRS with each requirement being fulfilled by either one module or the conjunction of several modules. For greater detail of the modules and what is needed to implement them there is the Module Interface Specifications (MIS), while the schedule of when these modules will be implemented is included in the Gantt and Pert charts.

### **2.2 Anticipated and Unlikely Changes**

#### **2.2.1 Anticipated Changes**

**AC1:** The specific hardware on which the software is running

**AC2:** The algorithm used for hand ranking

**AC3:** The deck shuffling algorithm

**AC4:** The card distribution methods

**AC5:** The pot/chip distribution methods

**AC6:** AI Opponent chip betting methods

#### **2.2.2 Unlikely Changes**

**UC1:** Input/Output devices (Input: File and/or Mouse, Output: File, Memory, and/or Screen).

**UC2:** The algorithm to calculate hand strength will always use the input from players hand and community card

**UC3:** The game will always end when a player runs out of chips

**UC4:** Any change to game state are always display to output devices

**UC5:** Algorithm used to calculate AI Opponents action are defined using

the parameters defined in the player and pot module

### 3 Module Hierarchy

This section provides an overview of the module design. Modules are summarized in a hierarchy decomposed by secrets in Table 1. The modules listed below, which are leaves in the hierarchy tree, are the modules that will actually be implemented.

**M1: TexasHoldEm**

**M2: Pot**

**M3: Game**

**M4: Hand**

**M5: Deck**

**M6: Player**

**M7: AIOpponen**

**M8: Card**

Table 2: Module Hierarchy

<b>Level 1</b>	<b>Level 2</b>
Hardware Hiding	
Behaviour-Hiding Module	TexasHoldEm Module Pot Module Game Module
Software Decision	Hand Module Deck Module Player Module AIOpponent Module Card Module

### 4 Connection Between Requirements and Design

The design of the system is intended to satisfy the requirements specified in the SRS. The system is decomposed into modules, and the connection between requirements and modules is listed in Table 3.

## **5 Module Decomposition**

### **5.1 Hardware Hiding Module**

Services: Includes programs that provide externally visible behavior of the system as specified in the software requirements specification (SRS) documents. This module serves as a communication layer between the hardware-hiding module and the software decision module. The programs in this module will need to change if there are changes in the SRS.

Secrets: The contents of the required behaviors.

Implemented By:

#### **5.1.1 Texas Holdem Module**

Services: Displays GUI

Secrets: GUI implementation

Implemented by: NLTP

#### **5.1.2 Pot Module**

Services: Keeps track of chips in pot, and distributes chips to players

Secrets: Method of distributing chips

Implemented by: NLTP

#### **5.1.3 Game module**

Services: Manages the state of the game

Secrets: How the game transitions between states

Implemented by: NLTP

### **5.2 Behaviour-Hiding Module**

Services: Includes data structure and algorithms used in the system that do not provide direct interaction with the user.



Secrets: The design decision based on mathematical theorems, physical facts, or programming considerations. The secrets of this module are not described in the SRS.

Implemented By:

### **5.2.1 Hand Module**

Services: Keeps track players cards, and calculates hand ranking

Secrets: Algorithm for calculating ranking of cards

Implemented by: NLTP

### **5.2.2 Deck Module**

Services: stores, shuffles and distributes cards

Secrets: Method of card distribution

Implemented by: NLTP

### **5.2.3 Player Module**

Services: lets user fold/bet

Secrets: Method of betting/folding

Implemented by: NLTP

### **5.2.4 AIOpponent Module**

Services: Lets AI fold/bet

Secrets: Algorithm for calculating whether to bet/fold

Implemented by: NLTP

### 5.2.5 Card Module

Services: object that represents playing cards

Secrets: Card Contructor

Implemented by: NLTP

## 5.3 Software Decision Module

# 6 MIS of TexasHoldEm Module

## 6.1 Exported Access Programs

Table 3: Exported Access Programs for TexasHoldEm Module

Name	In	Out	Exceptions
displayError()	int	-	Exception*
paint()	Graphic	-	-
save()	-	text file	FileNotFoundException
load()	text file	-	FileNotFoundException
actionPerformed()	ActionEvent	-	-

\* all possible exceptions will be included, including but not limited to, FileNotFoundException and NullPointerException

## 6.2 Interface Semantics

### 6.2.1 State Variables

### 6.2.2 Environment Variables

path:Location of the directory to be searched for when saving or loading the  
save file imagePath: Location of the directory to be searched for images of  
cards

### 6.2.3 Assumption

load() cannot be called without save() being called once before

### 6.2.4 Access Program Semantics

Main will populate the game window by instantiating the frame that is within TexasHoldEm module which will add all the components such as buttons, and uses paint() to populate the window with image of cards from environment variable imagePath and other simple graphics and colour for background. The save() will look for a save file in path. If it exists, it will overwrite it, if not, save() will create a new file. Load() will look for a save file in path and load all necessary data into the game and use Game module to modify the data for the affected modules. actionPerformed() will read the button input and call methods within Pot module based on its corresponding button.

displayError():

**Input** - Parameter of int type

**transition** - Displays an error corresponding to the input

## 7 MIS of Pot Module

### 7.1 Exported Access Programs

Table 4: Exported Access Programs for Pot Module

Name	In	Out	Exceptions
anti()	-	-	-
call()	-	-	-
raise()	int	-	-
fold()	-	-	-
getBet()	-	int	NullPointerException
getPot()	-	int	-
roundEndEvaluate()	-	-	-

### 7.2 Interface Semantics

#### 7.2.1 State Variables

pot:int

bet:int

player1:Player

player2:Player

### 7.2.2 Environment Variables

### 7.2.3 Assumption

### 7.2.4 Access Program Semantics

anti uses the Player module to modify both players' chips and also modifies the state variable pot by the same amount taken from player's chips. Call() and raise() uses the Player module to modify corresponding player's chips by the amount read by TexasHoldEm module. Fold() uses the Game module to modify the corresponding playerFolded parameter and also uses Player module to modify the amount of chips lost or gained. roundEndEvaluate() uses Player module to modify the amount of chips and uses the Game module to initialize a new round.

getPot():

**Input** - No input required

**Output** - Returns the value corresponding to the parameter pot

getBet():

**Input** - No input required

**Output** - Returns the value corresponding to the parameter bet

## 8 MIS of Game Module

### 8.1 Exported Access Programs

Table 5: Exported Access Programs for Game Module

Name	In	Out	Exceptions
playerSwitch()	-	-	-
deal()	-	-	-
nextCard()	-	-	-
isEndGame()	-	boolean	-
roundEnd()	-	-	-
player1Folded()	-	boolean	-
player2Folded()	-	boolean	-

## 8.2 Interface Semantics

### 8.2.1 State Variables

currentPlayer:int  
Deck: list of Cards  
player1: Player  
player2: Player  
endGame: boolean  
player1Fold: boolean  
player2Fold: boolean

### 8.2.2 Environment Variables

### 8.2.3 Assumption

### 8.2.4 Access Program Semantics

roundEnd() will call upon methods in Pot to evaluate, distribute the pot to the players and modify the parameters Deck, endGame, currentPlayer and each Player's hand by initializing the corresponding parameters as their default value or distribute new cards to the players.

playerSwitch():

**Input** - No input required

**transition** - modify currentPlayer between 2 values

deal():

**Input** - No input required

**Output** - Populates the Player hand variable

nextCard():

**Input** - No input required

**Output** - Modifies both players' hands by adding the same additional card to each hand

isEndGame():

**Input** - No input required

**Output** - boolean value corresponding to the value of parameter endGame

player(1/2)Folded():

**Input** - No input required

**Output** - boolean value corresponding to the value of their respective parameters

## 9 MIS of Deck Module

### 9.1 Exported Access Programs

Table 6: Exported Access Programs for Deck Module

Name	In	Out	Exceptions
setDeck()	-	-	-
Shuffle()	-	Deck	-

### 9.2 Interface Semantics

#### 9.2.1 State Variables

Deck: list of cards

#### 9.2.2 Environment Variables

#### 9.2.3 Assumption

Card module is created before Deck module

#### 9.2.4 Access Program Semantics

setDeck:

**Input** - No input required

**Output** - Populates the Deck with Cards

Shuffle: **Input** - No input required

**Transition** - Modifies the Deck so that it is randomized

## 10 MIS of Card Module

### 10.1 Exported Access Programs

### 10.2 Interface Semantics

#### 10.2.1 State Variables

rank:int

suit:char

Table 7: Exported Access Programs for Card Module

Name	In	Out	Exceptions
getRank()	-	int	DNE
getSuit()	-	char	DNE

### 10.2.2 Environment Variables

### 10.2.3 Assumption

### 10.2.4 Access Program Semantics

Get:

**Input** - No input required

**Output** - return value of corresponding parameter

## 11 MIS of Player Module

### 11.1 Exported Access Programs

Table 8: Exported Access Programs for Player Module

Name	In	Out	Exceptions
getChips()	-	-	-
gainChips()	int	-	-
loseChips()	int	-	EMPTY
getHand()	-	Hand	NullPointerException
setHand()	Hand	-	-

### 11.2 Interface Semantics

#### 11.2.1 State Variables

chips:int

Hand: list of Cards

### 11.2.2 Environment Variables

### 11.2.3 Assumption

getHand() can only be called after setHand()

### 11.2.4 Access Program Semantics

Get:

**Input** - No input required

**Output** - Each Get method will return value of the corresponding parameter

setHand:

**Input** - Input will take parameter Hand

**Transition** - Modifies the state of Hand

(gain/lose)Chips:

**Input** - Input will take parameter of int type

**Transition** - Modifies the state of chips

## 12 MIS of AIOpponent Module

### 12.1 Exported Access Programs

Table 9: Exported Access Programs for AIOpponent Module

Name	In	Out	Exceptions
getAction()	-	-	-

### 12.2 Interface Semantics

#### 12.2.1 State Variables

#### 12.2.2 Environment Variables

#### 12.2.3 Assumption

getAction is only called when it gets to its turn in the Game module



#### 12.2.4 Access Program Semantics

getAction will determine the move the AI will take by evaluating the current state of the board and its current Hand, and will call upon the Game module for its action.

### 13 MIS of Hand Module

#### 13.1 Exported Access Programs

Table 10: Exported Access Programs for Hand Module

Name	In	Out	Exceptions
evaluate()	Hand	int	-
getStrength()	-	int	-
myHand()	-	String	-

#### 13.2 Interface Semantics

##### 13.2.1 State Variables

hand: list of Cards

strength: int

##### 13.2.2 Environment Variables

##### 13.2.3 Assumption

evaluate() is only called when roundEnd() or player1Folded() or player2Folded() from game is called.

##### 13.2.4 Access Program Semantics

getStrength():

**Input** - No input required

**Output** - Returns value of parameter strength

myHand():

**Input** - No input required

**Output** - Returns the Cards in hand

evaluate():

## 14 Use Hierarchy

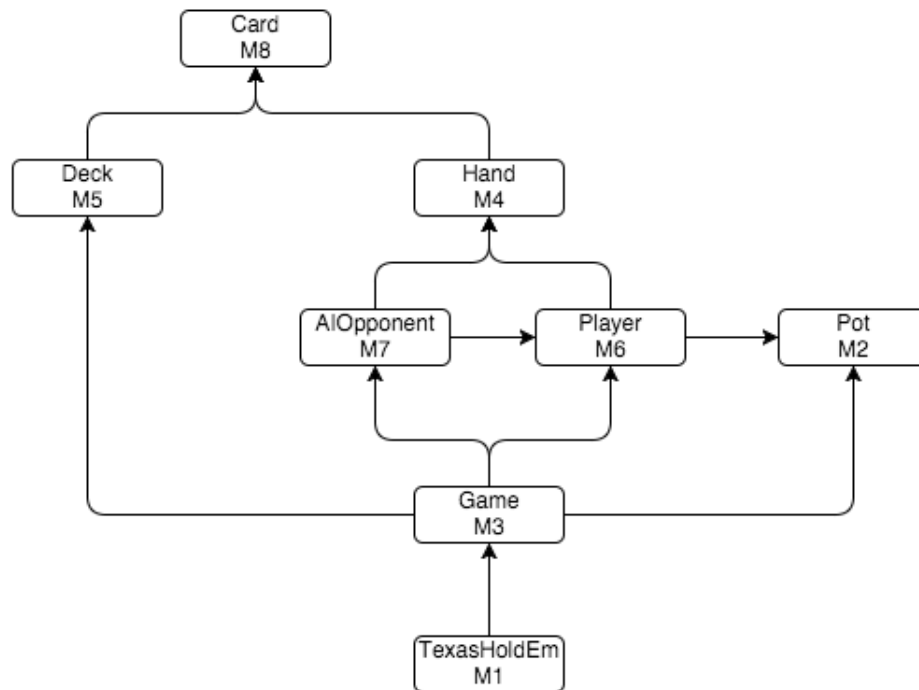


Figure 1: Use Hierarchy Diagram

## 15 Traceability Matrix

Table 11: Traceability Matrix Between the Module And Requirements

Requirements	Modules
R1	N/A
R2	M1
R3	M2,M6,M7
R4	M2,M3
R5	M6,M7
R6	M4,M8
R7	M3
R8	M2
R9	M2
R10	M2,M3

Table 12: Traceability Matrix Between Anticipated Changes And Requirements

Anticipated Changes(AC)	Modules
AC1	N/A
AC2	M4
AC3	M5
AC4	M8
AC5	M2
AC6	M7

## 16 Schedule