

**Faculty of Informatics and Computer Science**

Information System/Computer Networks/Software Engineering/ Computer Science

**Estimation Card Game Artificial Intelligence**

By: Ahmed Hatem Tageldin

Supervised By

Dr. Ayman Ghoneim

Abstract

Estimation is a popular trick-taking card game. There are several applications on the market. However, as most of them focus mainly on the multiplayer component of the game, the Artificial Intelligence component is sometimes neglected. The objective of this project is to create an Estimation Artificial Intelligence system that is capable of simulating the expertise and unpredictability of a real human player. There are other related works similar to Estimation, such as Whist. Whist is an English trick-taking card game, however it is internationally known. Therefore, resources are more readily available for it. Instead of starting from scratch when building the Artificial Intelligence system, similar existing card games can be analysed and their Artificial Intelligence may be extracted. It can be used as a guide or even a template for further refinements.

Attestation &Turnitin Report

I understand the nature of plagiarism, and I am aware of the University’s policy on this.

I certify that this report reports original work by me during my University project except for the following (*adjust according to the circumstances*):

* The technology review in Section 2.5 was largely taken from [17].

**Signature** *Ahmad Hatem* **Date** 15/12/2016

Table of Contents

[Abstract ii](#_Toc484285368)

[Attestation &Turnitin Report iii](#_Toc484285369)

[1 Introduction 1](#_Toc484285370)

[1.1 Overview 1](#_Toc484285371)

[1.2 Problem Statement 1](#_Toc484285372)

[1.3 Scope and Objectives 1](#_Toc484285373)

[1.4 Report Organization (Structure) 1](#_Toc484285374)

[1.5 Work Methodology 1](#_Toc484285375)

[2 Related Work 3](#_Toc484285376)

[2.1 Background 3](#_Toc484285377)

[2.1.1 Estimation Game Logistics 3](#_Toc484285378)

[2.2 Literature Survey & Analysis 7](#_Toc484285379)

[3 Proposed solution 9](#_Toc484285380)

[3.1 Object Oriented Programming 9](#_Toc484285381)

[3.2 Functional Requirements 9](#_Toc484285382)

[3.2.1 System Requirements 9](#_Toc484285383)

[3.2.2 Player Requirements 9](#_Toc484285384)

[3.3 Design 10](#_Toc484285385)

[3.3.1 Structural Model – Class Diagram 10](#_Toc484285386)

[3.3.2 Behavioural Model – Sequence Diagrams 12](#_Toc484285387)

[4 Implementation 17](#_Toc484285388)

[5 Testing and Evaluation 18](#_Toc484285389)

[6 Conclusions and Changes 19](#_Toc484285390)

[6.1 Summary 19](#_Toc484285391)

[References 20](#_Toc484285392)

[Appendix 1 21](#_Toc484285393)

[Appendix 2 22](#_Toc484285394)

[Appendix 3 23](#_Toc484285395)

[Appendix 4 24](#_Toc484285396)

[Appendix 5 25](#_Toc484285397)

[Appendix 6 26](#_Toc484285398)

Table of Figures

[Figure 1 - Scoring Table 6](#_Toc484285399)

[Figure 2 - startPlay Sequence Diagram 12](#_Toc484285400)

[Figure 3 - compareWeight Sequence Diagram 13](#_Toc484285401)

[Figure 4 - User openBidding Sequence Diagram 14](file:///C:\Users\Ahmad\Documents\GitHub\Estimation\%5bGP%5d%20Estimation%20Card%20Game%20Dissertation.docx#_Toc484285402)

[Figure 5 - Computer openBidding Sequence Diagram 15](file:///C:\Users\Ahmad\Documents\GitHub\Estimation\%5bGP%5d%20Estimation%20Card%20Game%20Dissertation.docx#_Toc484285403)

[Figure 6 - Computer dashCall Sequence Diagram 16](file:///C:\Users\Ahmad\Documents\GitHub\Estimation\%5bGP%5d%20Estimation%20Card%20Game%20Dissertation.docx#_Toc484285404)

[Figure 7 - Round deal Sequence Diagram 21](file:///C:\Users\Ahmad\Documents\GitHub\Estimation\%5bGP%5d%20Estimation%20Card%20Game%20Dissertation.docx#_Toc484285405)

[Figure 8 - Round secondRoundBids Sequence Diagrams 22](file:///C:\Users\Ahmad\Documents\GitHub\Estimation\%5bGP%5d%20Estimation%20Card%20Game%20Dissertation.docx#_Toc484285406)

[Figure 9 - Round startBidding Sequence Diagram 23](file:///C:\Users\Ahmad\Documents\GitHub\Estimation\%5bGP%5d%20Estimation%20Card%20Game%20Dissertation.docx#_Toc484285407)

[Figure 10 - User secondRoundBidding Sequence Diagram 24](file:///C:\Users\Ahmad\Documents\GitHub\Estimation\%5bGP%5d%20Estimation%20Card%20Game%20Dissertation.docx#_Toc484285408)

[Figure 11 - Session startSession Sequence Diagram 25](#_Toc484285409)

[Figure 12 - Session nextRound Sequence Diagram 26](file:///C:\Users\Ahmad\Documents\GitHub\Estimation\%5bGP%5d%20Estimation%20Card%20Game%20Dissertation.docx#_Toc484285410)

# Introduction

## Overview

Estimation is a four-player trick-taking card game. It is an Arab variation of the classic English card game Whist. The objective of the game is to estimate the amount of tricks, given a hand of thirteen cards. Each player plays one card in each hand. Whoever wins the hand collects a trick. There are thirteen tricks in each round. The person with the highest score at the end of eighteen rounds is the winner. The aim of the project is to create a playable Estimation game and simulate an artificially intelligent single player computer bot.

## Problem Statement

There are several Estimation card game applications on the market. Most applications focus their efforts on the online multiplayer side of the game, thus neglecting the single player side. However, most of them do not contain an Artificial Intelligence component that an experienced player would find sufficiently challenging.

## [Scope](http://www.cs.stir.ac.uk/~kjt/research/conformed.html) and Objectives

The objective of the project is to create an Artificial Intelligence component for the card game Estimation. By combining expertise in playing the game and passion in Java programming, the expected outcome is an Artificial Intelligence component that can simulate the thought process and the unpredictability of an experienced human player. The main expected deliverable is a Java application. However, seeing as Android development is based on Java, an Android application may be delivered as an extension deliverable if the project can be sufficiently completed in due time.

## Report Organization (Structure)

Section 2 covers the related works, their analysis and their potential usefulness to the project. Section 3 covers the proposed solution of the project, as to how the problem will be approached. Section 4 concludes the report with a summary of all the previous sections.

## Work Methodology

Plan A is to find projects fitting the same criteria and learn from their structure, their source codes and concepts. They can be used as guides. If such projects can be found, it will eliminate the need to start from scratch. For instance, other Estimation applications.

Plan B is to find projects fitting similar criteria and learn from their structure, their source codes and concepts. Although not ideal, it will provide a foundation on which to work from. For instance, variations of trick-taking games such as Whist.

Plan C is to find projects that share the same common basic attributes as Estimation. For instance, any card game application such as Poker, Hearts, Spades, etc.

Plan D is to start from scratch, which is the least ideal scenario as it involves much trial and error.

# Related Work

## Background

As per plan A, mentioned in section 1.5, the most ideal scenario is to find projects fitting the same criteria. Specifically, other Estimation projects. The main source of such related works is GitHub, which is a web-based repository service where developers can upload their projects and source codes online. The biggest problem with finding resources specifically for Estimation is that it is a region-specific card game, specifically in the Middle East. Therefore, resources are scarce, however, not non-existent.

The alternative to plan A, namely plan B, is to find projects fitting similar criteria. This may include other trick-taking games, such as Whist, Spades, Contract Bridge, Oh Hell, etc. The advantage to plan B is that these card games are international, therefore resources are much more likely to be readily available. The Artificial Intelligence component is likely to be more advanced as the game has a wider market, therefore more competition. The most useful part of finding resources is understanding various concepts of Artificial Intelligence programming, especially relating to card game science. Probabilities and statistics are likely to play an important role in the formulation of the Artificial Intelligence component.

### Estimation Game Logistics

#### Rules

The objective of the game is to estimate the amount of tricks, given a hand of thirteen cards. Each player plays one card in each hand. Whoever wins the hand collects a trick. There are thirteen tricks in each round. The card rankings are as follows, from highest to lowest: Ace, King, Queen, Jack, 10, 9, 8, 7, 6, 5, 4, 3 and 2.

#### Card Suits

There are four suits in a deck of cards. In Estimation, they are given a certain weight ranking. The ranking of the suits is as follows:

1. Spades
2. Hearts
3. Diamonds
4. Clubs

#### Game Structure

A game, or **Session**, of Estimation consists of eighteen rounds. The first thirteen rounds are played normally, with normal bidding rules. The final five rounds of the game are played in a fast bidding mode. The trump suit of each round is predetermined. The order of the trump suits in each of the final five rounds is as follows: Suns, Spades, Hearts, Diamonds and Clubs. Because the trump suits of the final five rounds are pre-determined, dash call may not be declared. If at any time a round is restarted, the score of the following round is incremented by two.

#### Round Structure

Each player receives a hand of thirteen cards. The objective of the game is to correctly estimate the amount of tricks that can be taken with the given hand. A round consists of thirteen hands; four cards for each hand, one from each player, totalling 52 cards. The trump suit is the suit that is elevated above its normal rank, which is chosen at the beginning of each round. At the beginning of each round, players take turns bidding for the trump suit.

#### Bidding

At the beginning of each round, players will be given a chance to declare a dash call. A dash call means that the player must not collect a single trick for the entire round. A dash call may only be declared before the trump suit bidding begins. The player on the right of the dealer begins bidding for the trump suit. A player may choose to make a bid or pass to the next player. The player with the highest bid may choose the trump suit. The bidder may also choose the trump suit as Suns, which means that there is no trump suit; the highest card of the played suit always wins. The trump suit ranking is as follows: Suns, Spades, Hearts, Diamonds and Clubs. For instance, a call of 5 Suns is greater than a call of 5 Spades, however a call of 6 Spades is greater than a call of 5 Suns. After the trump suit is chosen, the rest of the players declare their estimate bid, starting from the right of the trump suit bidder. The total sum of the players’ bids must not be equal to thirteen. The last player to make a bid estimate, or the player to the left of the trump suit bidder, has a bidding limit imposed on them. The round must either be under thirteen or over thirteen. If the last bidder makes a bid that is under or over the limit by at least two, they are flagged as a risk. If the round is under thirteen, that means that at least one player will take an extra trick. If the round is over thirteen, that means that at least one player won’t collect all of their estimated tricks. This system guarantees that at least one player will lose in each round.

#### Gameplay Flow

Once everyone has finished bidding, the trump suit bidder opens play. If the trump suit is Suns, the highest card of the played suit wins the trick. Otherwise, the highest card of the played suit wins the trick, unless someone interjects with a trump suit card. The card thrown by the player must be of the same suit. If a player does not have any cards of that suit, any card may be thrown, including trump suit cards. If more than one trump suit card is played, the highest trump suit card wins. In the fast bidding rounds, the player to the right of the dealer starts bidding in the pre-determined trump suit. The order of the trump suits in each of the final five rounds is as follows: Suns, Spades, Hearts, Diamonds and Clubs. A player wins the round if they collect the exact amount of tricks they bid at the beginning of the round.

#### Exceptions

A round may be restarted in two cases; the first of which is during call bidding, if every player chooses to pass instead of bidding. The second case is when every player loses the round. In other words, if no player collects the exact number of tricks they bid at the beginning of the round, the round is restarted. Whenever a round is restarted, the score multiplier for the following round is incremented by two.

#### Scoring Model

The scoring model was adapted from an Estimation Calculator application. [1]

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Case** | | | | **Scoring Equation** |
| Win | Caller / With | Call > 7 | Only winner |  |
| Not only winner |  |
| Call <= 7 | Only winner |  |
| Not only winner |  |
| Not Caller | Dash Call | Only winner |  |
| Not only winner |  |
| Normal Bid | Only winner |  |
| Not only winner |  |
| Lose | Caller / With | Call > 7 | Only loser |  |
| Not only loser |  |
| Call <= 7 | Only loser |  |
| Not only loser |  |
| Not Caller | Dash Call | Only loser |  |
| Not only loser |  |
| Normal Bid | Only loser |  |
| Not only loser |  |

Figure 1 - Scoring Table

#### Notes

* **With** refers to the case when a player bids a number of tricks equal to the caller. For instance, if the trump suit caller bids 5 Suns and a player bids 5 tricks, they are considered to be **With.** The scoring calculation for the caller and the **With** players is the same.
* The *call* variable refers to the amount of tricks the player bid.
* The *risk* variable refers to the amount of tricks the last bidder called. If a player bids 2 tricks under or over the limit, they are flagged as a risk. If a player bids 4 tricks under or over the limit, they are flagged as a double risk. If a player bids 6 tricks under or over the limit, they are flagged as a triple risk. Each risk is multiplied by 10 and added to the score. For instance, if a player bids 3 tricks under the limit, they are flagged as a double risk; when the score is calculated, 20 points are added to the score, before the multiplier.
* The *difference* variable refers to the absolute value of the predicted number of tricks subtracted from the actual number of tricks collected.
* If only one player wins, 10 points are added to the score calculation, before the multiplier. The same calculation applies if only one player loses.
* The *Multiplier* variable may be affected by several factors. If any of the following phenomena takes place, the*Multiplier* variable is incremented by two:

1. The previous round had no winners
2. The previous round had every player pass in the bidding phase
3. Three players called **With** the trump suit caller
4. Two players declare Dash Call in one round

## Literature Survey & Analysis

**Existing Estimation projects are scarce, ones with open-source code even more so. There exist three Estimation card games on the Google Play Store, the most relevant of which is Pocket Estimation.**

Projects similar to Estimation are much more likely to yield useful results. One such project is Whist. Whist is a classic English trick-taking card game originating from the 18th century. There exist several Android applications for the game Whist. The most popular of those applications is aptly named “Whist”. The creator of this game has uploaded the source code and the documentation of the game to GitHub. The original source code language is Java. All things considered, this is the main source that will be used for the program structure and architecture. The creator of this project is Nikos Tsaousis. [3] Within the project uploaded to GitHub, he has included the code architecture (Model View Presenter), the class diagram, the techniques used in the creation of the Artificial Intelligence computer opponents and the software design patterns. In comparison to the other Estimation projects, Nikos’s application is much more helpful as it is documented and explained. The only downside of this project is that it is very complex.

# Proposed solution

## Object Oriented Programming

As an expert Estimation player, the experienced gained over the years will be very useful in creating the Artificial Intelligence component. Naturally, as the program is developed, there will be several versions of the Artificial Intelligence with increasing complexity and difficulty as more time is spent on the code. Each version may be used as a benchmark to test against the final product. The game may have several difficulties, with the final version being the hardest; each version modelling a slightly increased difficulty with different tactics, strategies and approaches. The final version should be able to simulate the difficulty and unpredictability of playing against a real player.

Another proposed solution is one that actively learns from the player, gauging their level of expertise, evolving and adjusting its difficulty accordingly. However, this type of system is likely to be more complex because of the learning agent. It may use the advantage of having three computer opponents versus one human opponent. Thus, allowing it a much broader perspective of the game instead of the perspective of one opponent.

## Functional Requirements

### System Requirements

* Programming the game rules
* Programming the game structure
* Integrating exception handling
* Create bidding system to manage normal and fast bidding

### Player Requirements

* Manage score calculation model and display player scores
* Facilitate computer and user player management
* Create artificially intelligent computer bot capable of analysing the game and making decisions to maximise chances of success, which includes
  + Dash Call system
  + Bidding system

## Design

### Structural Model – Class Diagram

#### Abstract Classes

Abstract classes are to be used in the project. An abstract class must have at least one method declared as abstract, meaning they have no implementation. This forces any inheriting subclass to implement any abstract methods. Another feature of abstract classes is that they may not be instantiated, however they may be used in polymorphism. This is particularly useful as it allows the programmer to avoid making the distinction between the subclasses; instead, the programmer may use the general abstract superclass to refer to any subclass without limiting options. In particular, the Player class and the Suit class.

##### Player Class

The decision to make the Player class abstract came as a result of the varying implementations of its child classes. The User subclass represents human players, whose implementation requires active input from the user. For instance, a human player must decide on a call to bid on, thus entering a number and a Suit. On the other hand, the Computer subclass, which represents the artificially intelligent automated Player, has a different implementation than the User subclass. For instance, a Computer Player would automatically analyse the hand dealt to it and calculate the optimal bid in such circumstances without any prior input from the human Player.

##### Suit Class

The Suit class is a different case, as it serves a different function. It is also an abstract class, because its subclasses only have one element that varies. Each Suit has a set of cards which mark its domain. For instance, a Spades class’s domain is from the card 40 to the card 52. This behaviour is implemented in each class constructor. All the methods are inherited as they perform the same operation, but the numbers on which the operations are performed are what varies from subclass to subclass. The only exception to the Suit rules is the Suns class. Its domain is the whole deck of cards, from card 1 to card 52. That is because it does not allow a trump suit.

#### Relationships

* User-Player relationship – the Player class is specialised into a User class. This class is for human players. The User class overrides a number of methods for a more specific implementation. This is an example of inheritance.
* Computer-Player relationship – the Player class is specialised into a Computer class. This class is for computer bot players. The Computer class overrides a number of methods for a more specific implementation. This is an example of inheritance.
* Player-plays-Card relationship – a Player plays one Card at a time. However, over the course of one round, each player plays thirteen Cards. Therefore justifying the multiplicity: one Player plays many Cards.
* Player-joins-Session relationship – this relationship is an example of aggregation. A Player may exist outside of a Session. The object construction occurs outside of the Session class. A Player may be in only one Session at a time. Each Session is comprised of exactly four Players. Therefore justifying the multiplicity: four Players join one Session.
* Session-consists of-Round relationship – this relationship is an example of composition. The object construction occurs inside the Session class; a Round may not exist without a Session. A typical Session of Estimation is eighteen Rounds, however a Session may be ended early. One Round can only belong to one Session. Therefore justifying the multiplicity: one Session consists of one-to-eighteen Rounds.
* Player-plays-Round relationship – this relationship is an example of aggregation. A Player may exist outside of a Round. Each Player can only play in one Round at a time. A Round of Estimation must have four Players. Therefore justifying the multiplicity: four Players play one-to-many Rounds.
* Round-has-Call relationship – one Round has four Calls, one Call for each Player. The main Call for the Round is in a variable of its own, named *call*, while the rest of the Calls are stored in an ArrayList. Therefore justifying the multiplicity: one Round has four Calls.
* Player-makes-Call relationship – every Round, each Player makes one Call. Therefore justifying the multiplicity: one Player makes one Call.
* Call-of type-Suit relationship – one Call can be of one Suit type. Therefore justifying the multiplicity: one Call of type one Suit.

### Behavioural Model – Sequence Diagrams

The project will be dependent on many methods and functions, therefore only the most fundamental and important methods will be listed below.

#### Round – startPlay

Figure 2 – Round **startPlay (int cursor)** Sequence Diagram

**Explanation**

This method is responsible for the distribution of play during each round. After call bidding and second round bidding has taken place, the Round class initiates play. The cursor parameter is used to determine the person who opens play. The player who won the trump suit bid opens play, then the players take turns to play a card. Each card played is added to an ArrayList, in order, to keep track of the cards played in each hand. After four cards have been played, the winner of the hand is determined and that player has their tricks incremented. Then the hand is cleared to make space for the next four cards. This process is repeated thirteen times, as each player has thirteen cards, totalling 52 cards altogether. This method has no return.

#### Suit – compareWeight

Figure 3 –Suit **compareWeight (ArrayList <Integer> hand)** Sequence Diagram

**Explanation**

This method serves to be one of the most important functionalities in the system. It is responsible for comparing a set of four cards together and returning the largest card. The Round class calls this method when determining the hand winner after each hand. The first step is to sort the given hand, this is to reduce the potential amount of processing that the main algorithm has to do. As it is a list of integers, it can be sorted using the default **Collections.sort** comparator. The main algorithm has a series of if conditions with the method **trumpCheck(int)**, which returns a true value if the given integer is in the Suit object’s card domain. (See Section 3.3.1.1.2 Suit Class). For instance, the card 51 (King of Spades) would be in the domain of the Spades Suit. If all of the cards in the hand are in the domain of the object Suit, then the highest card wins. If not all of the cards in the hand are in the domain of the object Suit, then only the cards in the domain are compared. When this method is called in the Round object, the person who plays the first card in the hand is the one who decides the Suit object whose **compareWeight** method is called. This guarantees that there would be at least one card that fits the **trumpCheck** condition. For instance, if the first card in the hand is the Ace of Spades, the Spades **compareWeight** method gets called. There is only one exception to this rule, which is if one of the following players (not the first player) does not have a card of the played Suit and interjects with a trump card (based on the Round trump suit), the Round object calls the **compareWeight** method of the trump suit. This would mean that only the trump card played would be in the domain, meaning that it would be the highest card. This method returns an integer, the highest card.

#### User – openBidding

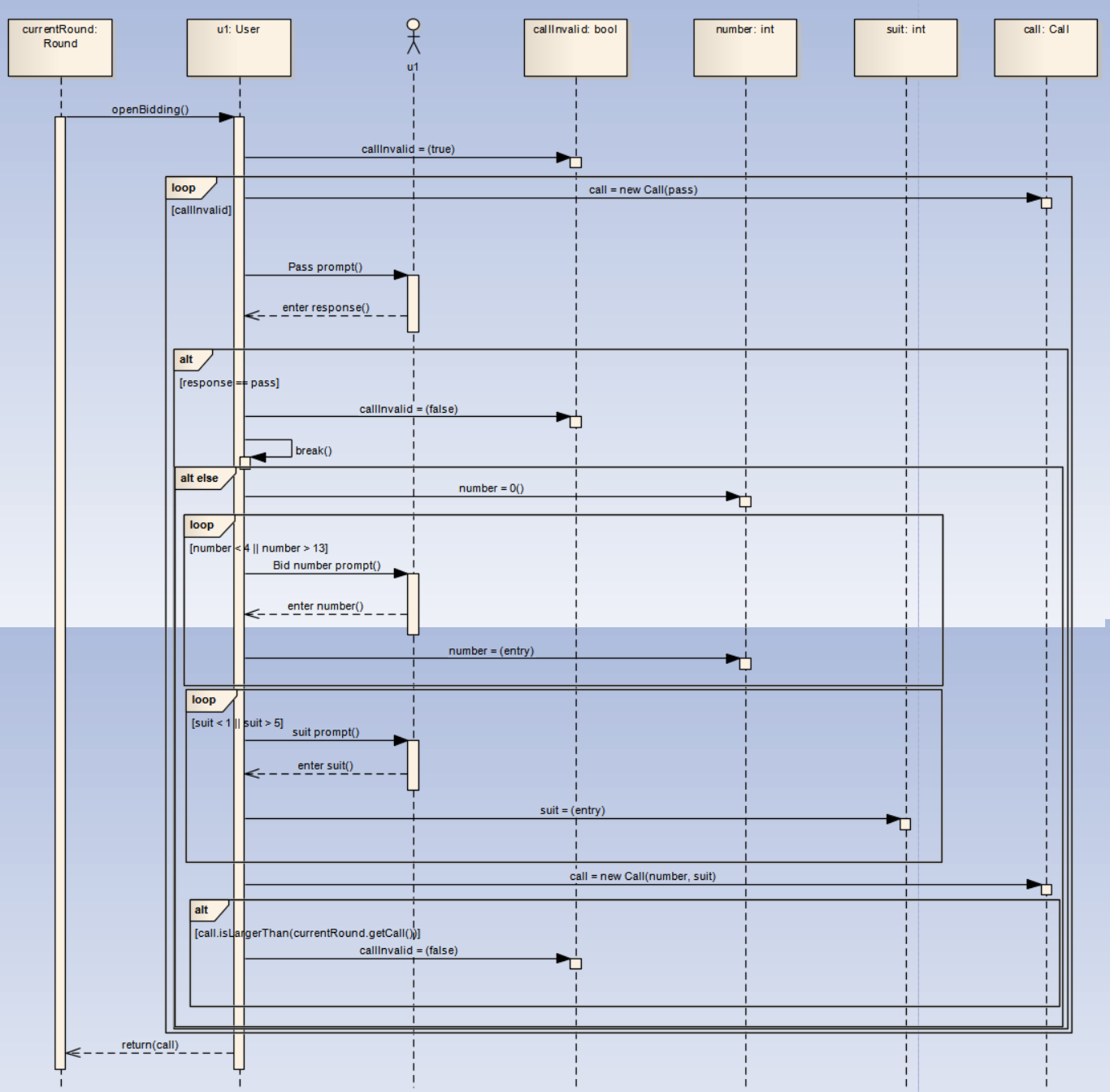


Figure 4 - User **openBidding()** Sequence Diagram

**Explanation**

This method allows human user Players to participate in the trump Suit bidding round. The Round object calls this method. The main while loop has a condition that forces the user to either pass or make a valid call. As long as the call is invalid, the loop will never break. The user is first allowed to pass on bidding, which would break the loop and result in a “Passed” call. Otherwise, the user is then directed to choose a number to bid between four and thirteen, as the trump Suit call cannot be less than four or greater than thirteen. After the number has been chosen, the user is then prompted to choose a Suit to bid on. The call is then created and compared with the biggest trump Suit call made so far in the Round. If it is bigger, then the loop is broken. This method returns the call made by the user.

#### Computer – openBidding

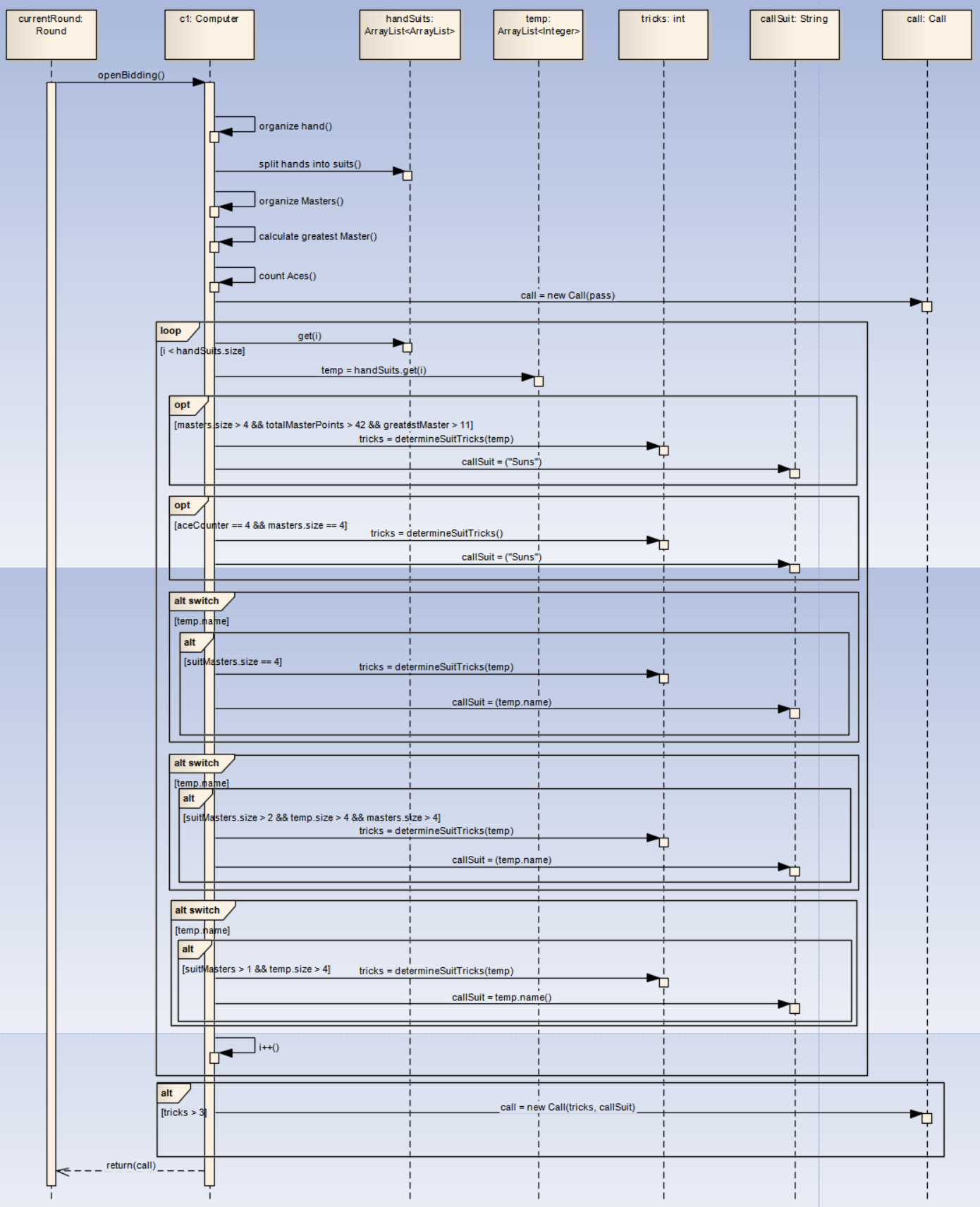


Figure 5 - Computer **openBidding()** Sequence Diagram

**Explanation**

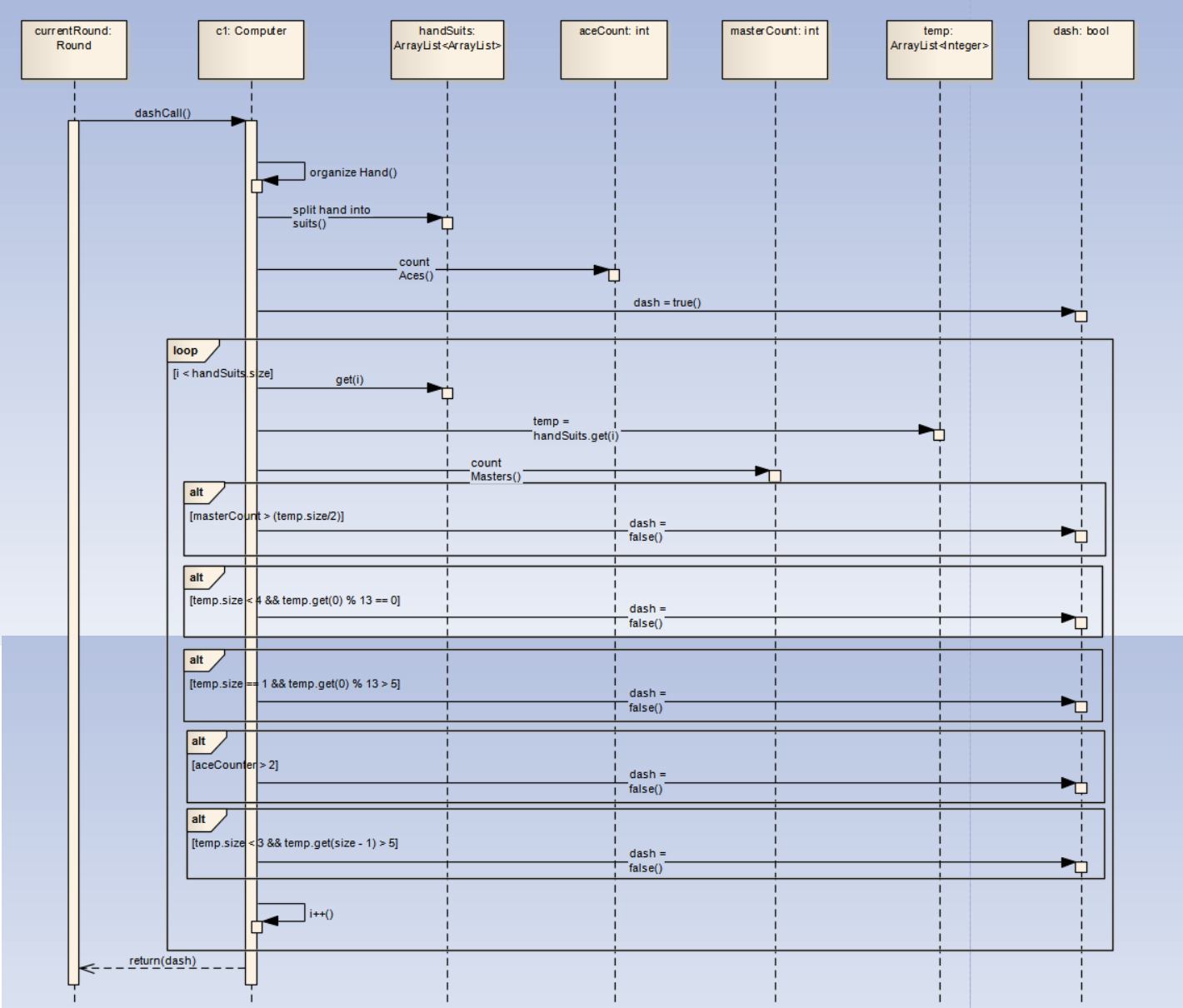
This method allows the computer Players to participate in the trump Suit bidding round. The Round object calls this method. The first step is to organize the hand into several lists and variables. Some organizing operations include counting the Aces in the hand, counting the masters, counting total master points, putting the cards in the hand into their respective Suit lists, putting masters into their respective master Suit lists and determining greatest master in each Suit. After organizing the hand, a loop traverses the Suit list and each Suit is subjected to five cases, each with specific criteria which must be satisfied:

1. Total amount of masters in hand is greater than 4  
   Sum of master points in hand greater than 42  
   Greatest master in hand higher than a Queen  
   *Call Suit = Suns*
2. Hand contains a total of 4 Aces  
   *Call Suit = Suns*
3. Suit contains a full house (Ace, King, Queen and Jack)  
   *Call Suit = Suit (Spades/Hearts/Diamonds/Clubs)*
4. Suit has more than 2 master cards  
   Suit has more than 4 cards  
   Hand has more than 4 master cards in total   
   *Call Suit = Suit (Spades/Hearts/Diamonds/Clubs)*
5. Suit has 2 or more master cards  
   Suit has 5 or more cards   
   *Call Suit = Suit (Spades/Hearts/Diamonds/Clubs)*

In each case, the tricks are counted by calling the method **determineSuitTricks (suit)**. The result is then put into a variable *tricks* and the call Suit is put into a variable *callSuit*. After the case loop, the method then checks if the *tricks* variable is greater than 3, as the smallest possible trump Suit call is 4. If it is greater than 3, then the call is made using the *tricks* and *callSuit*. Otherwise, the call is a “Passed” call. The method then returns the call generated.

#### Computer – dashCall

Figure 6 - Computer **dashCall()** Sequence Diagram



**Explanation**

# Implementation

# Testing and Evaluation

# Conclusions and Changes

## Summary

Estimation is a popular Arabic card game. However, it being region-specific poses a problem in finding resources for it. An alternative solution to that problem is to find similar games, such as Whist, and using it as a guide to help create an Artificial Intelligence component.

References

1. Mustafa Haddara, *Estimation, cmdline implementation of a card game* <https://github.com/MustafaHaddara/estimation>, September 2016.
2. Bramdemoor, *Estimations card game in ruby*<https://github.com/bramdemoor/Estimations>, December 2014
3. Nikos Tsaousis, Whist is a trick-taking, real-time multiplayer card game  
   <https://github.com/tsanikgr/whist>, December 2015

# Appendix 1

Figure 7 - Round deal Sequence Diagram

# Appendix 2

Figure 8 - Round secondRoundBids Sequence Diagrams

# Appendix 3

Figure 9 - Round startBidding Sequence Diagram

# Appendix 4

Figure 10 - User secondRoundBidding Sequence Diagram

# Appendix 5

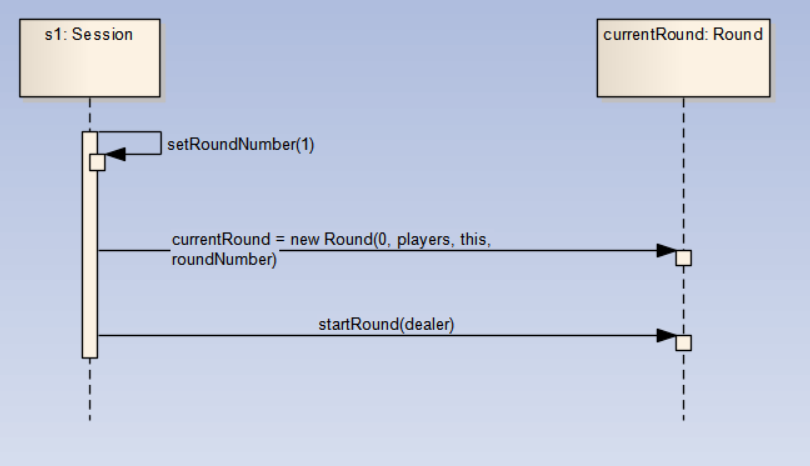


Figure 11 - Session startSession Sequence Diagram

# Appendix 6

Figure 12 - Session nextRound Sequence Diagram