Team Automaton



Software Design

Foundations of Software Engineering (EN.605.401.71. FA14)

Authors: Henoke Shiferaw, Shambhavi Shankrit, Ethan Wilansky

Contents

[Introduction 3](#_Toc403315497)

[Purpose & Scope 3](#_Toc403315498)

[Definitions, Acronyms, and Abbreviations 3](#_Toc403315499)

[References Overview 3](#_Toc403315500)

[Activity Diagrams 4](#_Toc403315501)

[Suggestion 4](#_Toc403315502)

[Suggestion from Other Player 5](#_Toc403315503)

[Accusation 6](#_Toc403315504)

[System Object Model 7](#_Toc403315505)

[Subsystems 7](#_Toc403315506)

[Object Descriptions 7](#_Toc403315507)

[Object Collaboration Diagrams 8](#_Toc403315508)

[Dynamic Model 10](#_Toc403315509)

[Sequence Diagram 10](#_Toc403315510)

[State Diagram 11](#_Toc403315511)

[Data Flow Diagram 12](#_Toc403315512)

[Supplementary Documentation 12](#_Toc403315513)

[Tools Used to Create Diagrams 12](#_Toc403315514)

[References 12](#_Toc403315515)

# Introduction

This software design document contains a set of UML diagrams that introduce the code design and behaviour of the Computer or Autonomous player in the game of Clue-Less.

## Purpose & Scope

This documentation provides a set of architecture diagrams for the Computer Player subsystem for the Clue-Less game. Design elements include the following diagrams:

* UML Class Diagrams
* Activity Diagrams
* Sequence Diagrams
* State Diagrams
* Data Flow Diagrams

The UML diagrams (Nishadha 2012) show the composition of the Computer Player classes and how a Computer Player object responds to inputs (public methods) made by a caller. The intended caller in this case is the Clue-Less server subsystem. Specific public method and attribute details are described in the Interface Specification while functional and non-functional requirements are described in the Requirements Document, neither of which are repeated here.

## Definitions, Acronyms, and Abbreviations

Computer Player – synonymous with the AI Player or Autonomous Player mentioned in other documentation for this project. The Computer Player is an object derived from a Python class that is able to take turns and be updated by a calling subsystem.

## References Overview

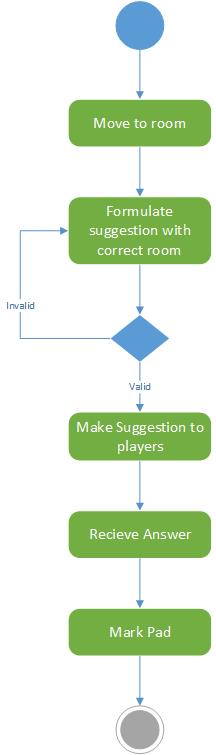
The citations section at the end of this document outlines any references used to create the content in this documentation. We used an OASIS software design outline specification (McElreath 2007) to develop the sections in this document. Sections covered in other project submissions have been removed from this document.

# Activity Diagrams

The following 3 activity diagrams present a high level representations of actions taken by the autonomous player during various stages during the Clueless game. These include, making a suggestion, interpreting suggestions from another player, and formulating an accusation.

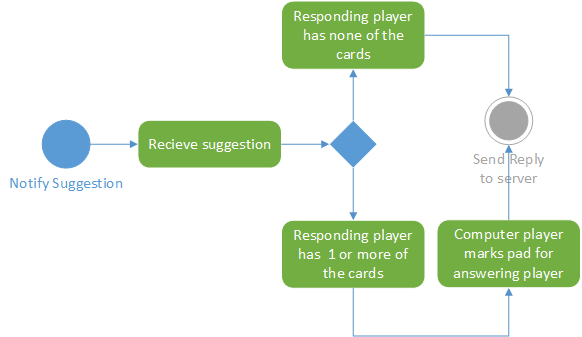
## Suggestion

Figure 1. Activity diagram showing a suggestion from an autonomous player



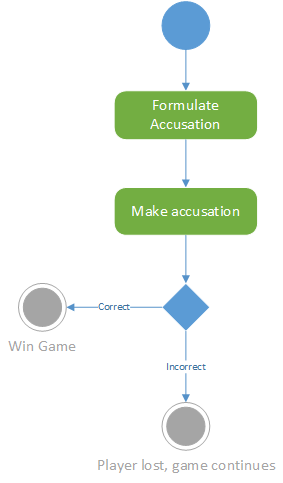
## Suggestion from Other Player

Figure 2. Activity diagram showing an autonomous player responding to a suggestion from another player



## Accusation

Figure 3. Activity diagram showing an accusation from an autonomous player



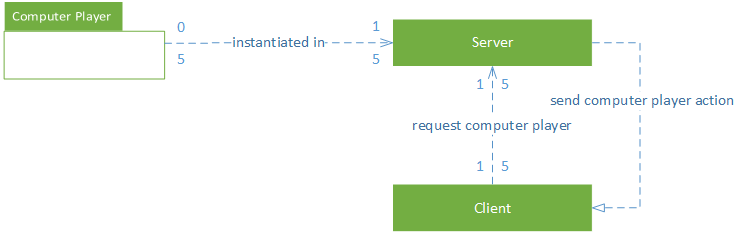
# System Object Model

There are three sub-systems in the Clue-Less game, Server, Client and Computer Player. This has been outlined in prior documentation. Therefore, this section simply outlines the overall view of the three sub-systems. The interface between the Computer Player sub-system and the Server is via a Python module that allows Player class instantiation in server memory.

## Subsystems

The following diagram shows a high-level relationship between the sub-systems. Emphasis by way of object shape is placed on the Computer Player for this design document.

Figure 4. High-level sub-system overview



# Object Descriptions

Player – core class from which the caller will instantiate computer player objects, one for each player.

Board – class that acts as a map of the Clue-Less board. Player uses this graph data structure to determine possible moves from its current position.

Pad – class that contains sub-tables used for tracking each player’s answers to suggestions. The Pad object contains a sub-table for each player in the game. Each Player gets a pad when the Player is instantiated.

# Object Collaboration Diagrams

The following two class diagrams show the classes and connections of the Computer Player. A few key dependencies are also shown in the class diagrams. The first diagram does not shows a high-level view of the classes and their associations and dependencies. The second diagram contains all of the public and private attributes and operations and class associations.

Figure 5. Summary View of the Computer Player Classes

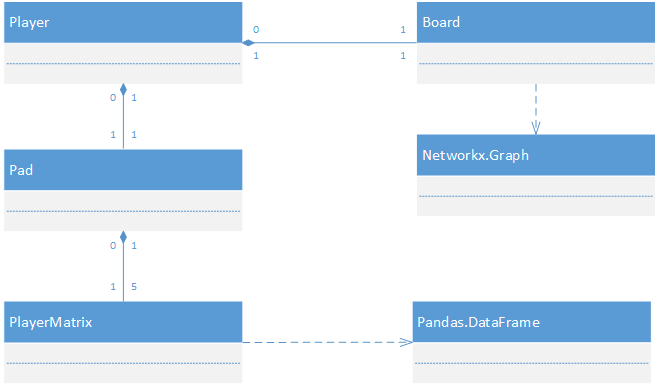
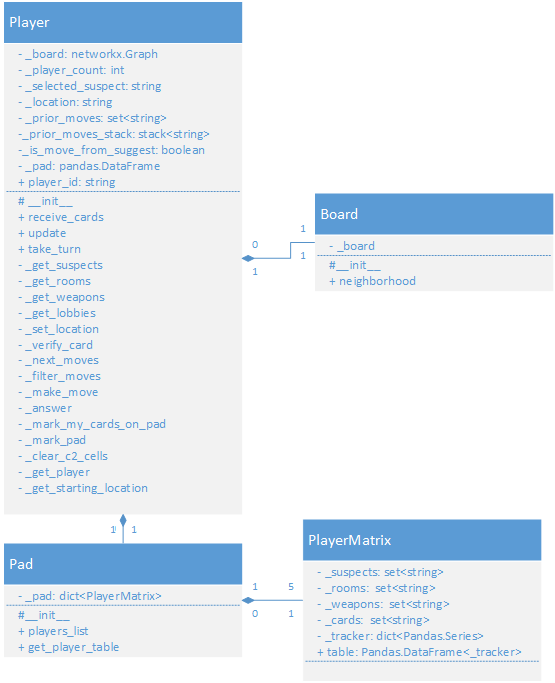


Figure 6. Detail view of the Computer Player Classes



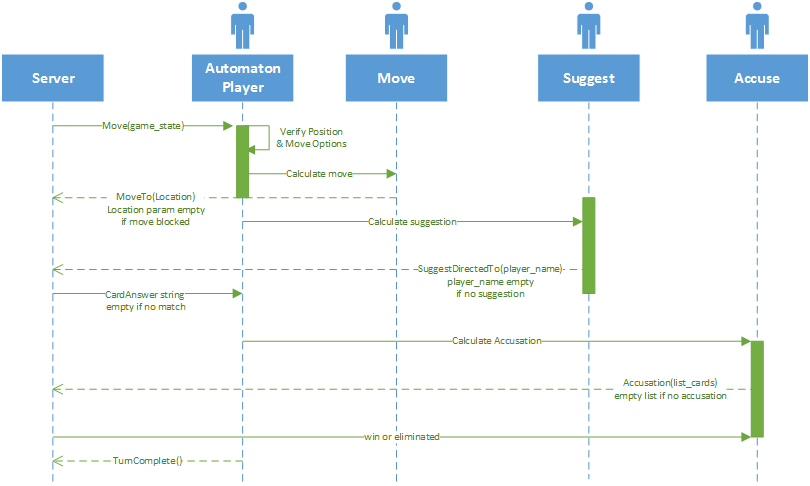
# 

# Dynamic Model

## Sequence Diagram

The sequence diagram for the autonomous player shows one of the more complex sequences that must be coded, making a move.

Figure 7. Autonomous Player Move Sequence Diagram

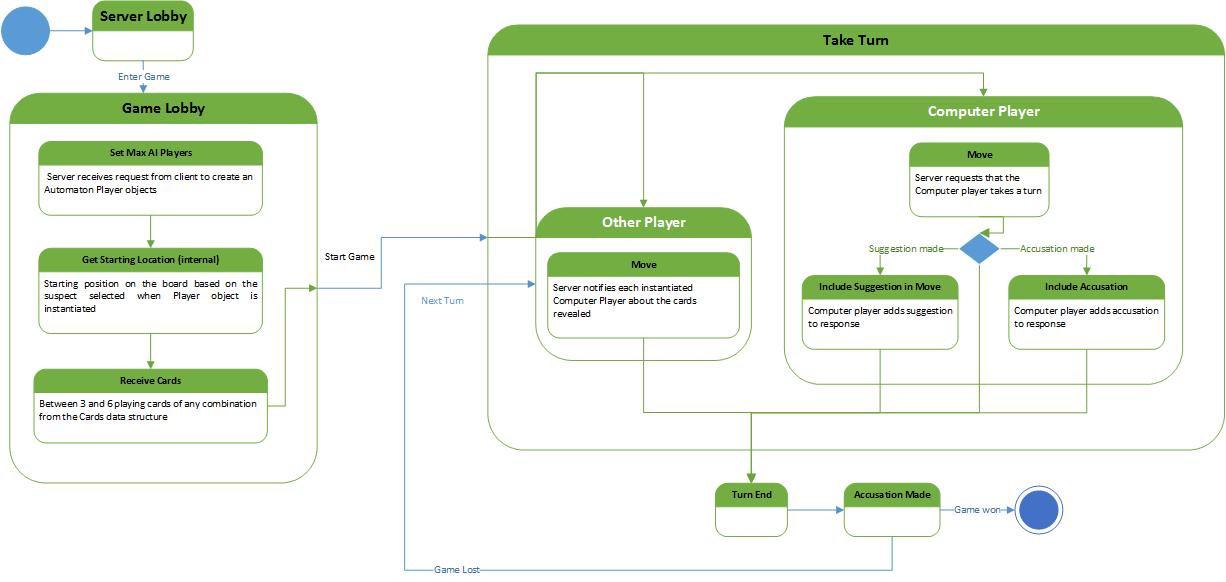


## State Diagram

The state transition diagram shows the different states in which an Autonomous player will be and the transition between these states.

The count of the Autonomous players playing the game, their starting position and the set of cards received by each one of them are the three states in the Game lobby. Next state is to make a move and then make an accusation or suggestion or do nothing. This takes the player to the state where the turn ends for this round of the game and in the next state the player are notified about the accusations made. If the accusation is right the player wins and the game ends, otherwise he/she loses the game and can only make suggestions in the next turns.

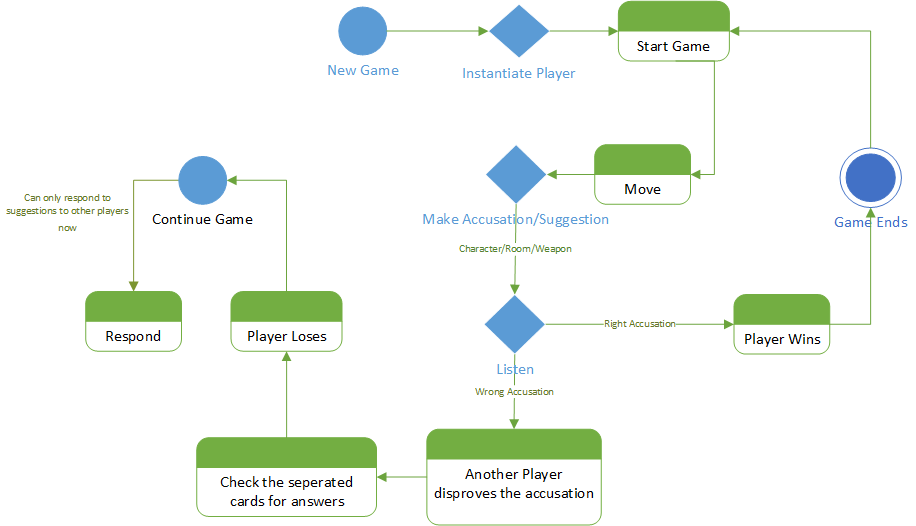
Figure 8. Take turn state transition



## Data Flow Diagram

The data flow diagram represents the perspective of an Autonomous player. It includes the interface between the different components beginning with the start of a new game to losing or winning the game (Pressman 2007).

Figure 9. Data flow diagram for autonomous player



# Supplementary Documentation

## Tools Used to Create Diagrams

The team used Microsoft Visio 2013 to create all of the diagrams in this specification. In addition, the class diagrams were initially generated from the code using pyNsourceGui32, an open-source python diagram generator. However, the diagrams weren’t complete. For example, associations were shown, but multiplicity was missing and there was nowhere to add it.

# References

McElreath, Rex. 2007. “XML Legal Document Utility - Example of a Software Design Specification.” *OASIS.* 20 04. https://www.oasis-open.org/committees/download.php/24846/Example-SoftwareDesignDocument-LegalXMLUtility.pdf.

Nishadha. 2012. “UML Diagram Examples.” *The Creately Blog.* 02 02. http://creately.com/blog/diagrams/uml-diagram-types-examples/#ClassDiagram.

Pressman, Robert. 2007. *Software Engineering: A Practitioner's Approach.* New York, New York: McGraw-Hill. Accessed November 2, 2014.