

Hanabi PRD

Mels Kolozyan

Xudong Li

Xufeng Lin

Evan Semenoff

Tiandu Xie

Contents

I	Introduction	6
1	Purpose	6
1.1	What is the Purpose of this Document?	6
2	Scope	6
2.1	Full Software Description	6
2.2	Software Features and Limitations	7
2.2.1	Client Features	7
2.2.2	Client Limitations	7
2.2.3	AI Features	7
2.2.4	AI Limitations	8
3	Definitions	8
3.1	Vocabulary	8
3.2	Acronyms	9
3.3	System Overview	9
3.4	Documentation Overview	9
4	References	10
4.1	Textbooks and Papers	10
4.2	Technical Documentation	10
II	Overall Description	11
1	Product Functions	11
1.1	Actors	11
1.2	Use Cases and Operating Concepts	11
1.2.1	Action Definitions	13
1.3	Use Case Diagram	15
1.4	Data Flow Diagram	16
2	User Characteristics	16

3	Constraints	17
3.1	Platform	17
3.2	Network Protocols	17
3.3	Timeframe	17
4	Assumptions and Dependencies	17
4.1	Libraries	18
III	Specific Requirements	19
1	External Interfaces	19
1.1	User Interface	19
1.2	Hardware Interface	20
1.3	Communications Interface	20
2	Functions	20
2.1	Validity Checking Inputs	20
2.2	Sequence of Operations	21
2.3	Responses to Abnormal Situations	23
2.3.1	Network Communication Errors	23
2.3.2	Error Handling and Recovery	24
3	Performance Requirements	24
3.1	Number of Users	24
3.2	Information Handling	25
4	Adaptability Requirements	25
IV	Verification Requirements	26
1.1	User Interface	26
1.2	Hardware Interface	26
1.3	Communications Interface	26
2	Functions	26
2.1	Validity Checking Inputs	26
2.2	Sequences of Operations	27
2.3	Responses to Abnormal Situations	28

2.3.1	Overflow and Underflow	28
2.3.2	Network Communication Errors	28
2.3.3	Error Handling and Recovery	28
3	Performance Requirements	28
3.1	Information Handling	29
4	Maintainability Verification	29
5	Reliability Verification	30
V	Appendices	30
6	Appendix A: Diagrams and Figures	31
7	Appendix B: Comparison between this Document and the ISO standards	36

Executive Summary:

Hanabi is a cooperative, solitaire-like game based around logical inference. Players give hints to help each other reason about hidden information, and use a combination of deductions, assumptions, and implications in an attempt to complete the game together.

This document outlines the requirements for the implementation of a game client by which players can enjoy the game of Hanabi over the internet. The goal is to make an efficient, aesthetic, and well-featured piece of software that captures the original spirit of the card game it was based on.

Part I

Introduction

1 Purpose

1.1 What is the Purpose of this Document?

This document describes the full requirements necessary for a client-server based implementation of the card game Hanabi. It follows (as much as applicable) the standards set out by the IEEE to address the concerns of what the client will be capable of, what is required by it, and how those concerns will be addressed.

In addition to the requirements of the software from a functional perspective, it also outlines the verification requirements that the software will be subject to later in development.

2 Scope

2.1 Full Software Description

The general scope of the software is split into two distinct modes:

Client Mode: This mode is the main 'game' mode of the software. In general, it allows a user to play Hanabi through a GUI based client. In this mode a player will be able to both create, join, and play games of Hanabi. Additionally, a basic tutorial will be available to explain the rules of the game. It's important to note that this mode is not responsible for the implementation of the game, it is responsible strictly for the presentation of the current game state, allowing the user to interact with it.

AI Mode: This mode will allow a user to connect a computer player to an existing game. It is to be implemented to allow execution directly from the command line. After being sent to a game, the AI will play until the game is completed.

2.2 Software Features and Limitations

2.2.1 Client Features

- a) The player will be given a simple, informative GUI that is both intuitive and clutter-free. The UI will focus heavily on helping the player keep track of the game state by allowing them to mark inferences on their cards. It will also guide the player by highlighting legal moves during gameplay, with the goal of making the game more accessible.
- b) In order to aid in such accessibility, the game will also feature some basic animations to help the player keep track of the game state. Two types of animations that will be important will be:
 - i. Animations that help the user understand what they're interacting with in the UI by doing things like highlighting buttons that are being hovered over, shifting objects when the cursor is placed over-top, or animating buttons when they're clicked on.
 - ii. Animations that help the user understand the state of the game. Since information conveyance is so important in Hanabi, these will have to be clear.
- c) Simple sounds like clicking and shuffling noises that give a player clear audio feedback when an action has been performed.

2.2.2 Client Limitations

- a) Although the game will have a GUI and basic animations, the general art design of the game will be kept minimalistic to fit within the time constraints of the project.
- b) The in-game client options should be kept as minimal as possible, although the intention is to add a small set of utility functions to the player, too many would not only obfuscate the clarity of the interface but also add unnecessary complexity to the program.

2.2.3 AI Features

- a) The AI player should be able to interact meaningfully with any game it is told to connect to and play a game from start to finish without causing a timeout.

- b) The AI player should perform objectively better than an AI which simply generates a random move and plays it if it's legal.

2.2.4 AI Limitations

- a) The AI player will have relatively low resource usage. The client should will run any pre-built computer produced after 2010.
- b) The AI player will not play a perfectly optimal move at all times, it doesn't need to perfectly solve the entire game of Hanabi.

3 Definitions

3.1 Vocabulary

Action A specific movement depended on different actors

Actor A role in the whole system, including clients and the server

Bug Error or failure of the program

Fuse A type of game tokens to indicate remaining mistakes allowed

Heuristic Evaluation system providing ideas how the artificial intelligence will choose a move

Library A collection of code invoked to construct the program

Packet A data packet, the smallest unit of internet data transferred between servers and clients

Protocol Rules and regulations for internet networking

Server Responds the request from clients, it sets games up and run all processes for programs interacting with the clients

3.2 Acronyms

AFK	Away from keyboard
BOK	Back on keyboard
FPS	Frames per second
GM	Game master
GUI	Graphical user interface
IEEE	Institute of electrical and electronics engineers
IP	Internet protocol
J2SE	Java 2 standard edition
JVM	Java virtual machine
TCP	Transmission control protocol
UI	User interface

3.3 System Overview

The system, in full, should comprise of two main functional components.

The Client The component that allows an end user to play a game of Hanabi with a group of other users over a network connection.

The AI A program that can be run which mimics the behaviour of a player after connecting with a server.

3.4 Documentation Overview

This document is organized to outline the basic requirements of the system by starting off with generalized, abstract requirements, and refining them to down to their more specific, and concrete parts.

Section 2.0 is concerned with the overall description of the system to be delivered, and will provide an outline of the actors and their use cases, the

constraints around which the system will be designed, and finally the set of hardware and software requirements for the end user.

Section 3.0 addresses in more detail, the use cases outlined in section 2.0, and expands these use cases into interfaces, functional, performance, and maintainability requirements.

Finally, section 4.0 addresses how the requirements of section 3.0 will be verified in a reliable, quantifiable way.

4 References

4.1 Textbooks and Papers

- David C. Kung; Object-Oriented Software Engineering: An Agile Methodology; McGraw-Hill, 2013
- Olivieri, J. (2009). Quantifying Software Reliability and Readiness. Retrieved from MITRE: <http://www.asq509.org/ht/a/GetDocumentAction/i/46491>
- Steve McConnell; Code Complete 2; Microsoft Press, 2004

4.2 Technical Documentation

- Dutchyn, C. (2019, January 22). Intermediate Software Engineering Assignment 1Requirements.
- Systems and software engineering Life cycle processes Requirements engineering. (2011, December 01). International Standard. Switzerland : ISO/IEC/IEEE.
- IEEE Recommended Practice for Software Requirement Specifications. (1998, October 20). IEEE Computer Society. Software Engineering Standards Committee.

Part II

Overall Description

1 Product Functions

This part outlines the overall requirements of the program, it covers the main users (actors) and their interactions with the software. It addresses the use cases for said actors and outlines a generalized data flow for the software. Finally, it will define the constraints model that these requirements will fall under.

1.1 Actors

There are three actors in this program:

1. A game master (GM) is the creator of the table and is responsible for specifying the number of players (including themselves) that will be playing. The action of joining a table is not possible for a GM, because as soon as they request the creation of a new table they automatically join this new table. Other than the pre-game setup a GM's capabilities and possible actions are identical to those of a regular player.
2. A regular player's available pre-game action is joining a game, which includes connecting to an already created server through a game-ID.
3. An AI bot is invoked by a human and given unique identifying information along with a game-id, which enable it to join the desired server. Upon joining, the AI is expected to play until the game is terminated.

1.2 Use Cases and Operating Concepts

With the actors identified, this section will focus on outlining their basic interactions with the software by specifying use cases for each actor, and then give a brief overview of what these actions entail.

1. GM Actions:
 - Create table

- Play a card
- Give a hint
- Discard a card
- View discarded cards
- Create personal notes for each card
- Start the AI (AFK)
- Stop the AI (BOK)
- Exit table

2. Regular Player Actions:

- Join table (player)
- Play a card
- Give a hint
- Discard a card
- View discarded cards
- Create personal notes for each card
- Start the AI (AFK)
- Stop the AI (BOK)
- Exit table

3. AI:

- Join table (AI)
- Play a card
- Give a hint
- Discard a card
- View discarded cards

1.2.1 Action Definitions

Create Table When the user is performing this action, they are adopting the 'Game Master' roll.

This action allows a player to create a table. They will do so by providing some basic information to the client such as:

- a) Number of players
- b) Time limit for making a move
- c) Server name

Join Table This is the basic action which allows either a human player or a computer player to join a table. Although they're essentially the same thing from the point of view of the server, they have slightly different usages.

- a) (player) A player can join a server by specifying the game-ID. After this they are able to do all of the list in-game actions.
- b) (AI) A human invokes the AI with a game-ID and identifying token to join a server via the command line.

Play Card This is the basic action of playing a card. After selecting a card to play, the card is then placed on the table face-up. It is then moved to either the discard pile or the play area depending on the validity of the move. If the move was invalid, the card is moved to the discard pile and the number of fuses decreases by one. This action consumes a turn, all players are informed of the play and gameplay resumes with the next player.

Give Hint This is the action which allows one player to communicate information with another player. The hint may be about either the colour or the number of the card(s), but not both. This action consumes a turn, all players are informed of the play and gameplay resumes with the next player.

Discard This action allows a player to discard a card from their own hand if the team does not have the maximum amount of information tokens. The player will select a card and then select the discard option. The selected card will be added to the discarded cards pile, and one information token will be restored. This action consumes a turn, all players are informed of the play and gameplay resumes with the next player.

View Discards At any point during the game, it is possible to view all the cards currently in the discard pile. Selecting this option during gameplay will bring up a graphical list of cards for the player to examine.

Create Notes This function will allow players to keep track of their own inferences about their cards by giving them an interface which allows them to record either an assumed number or colour.

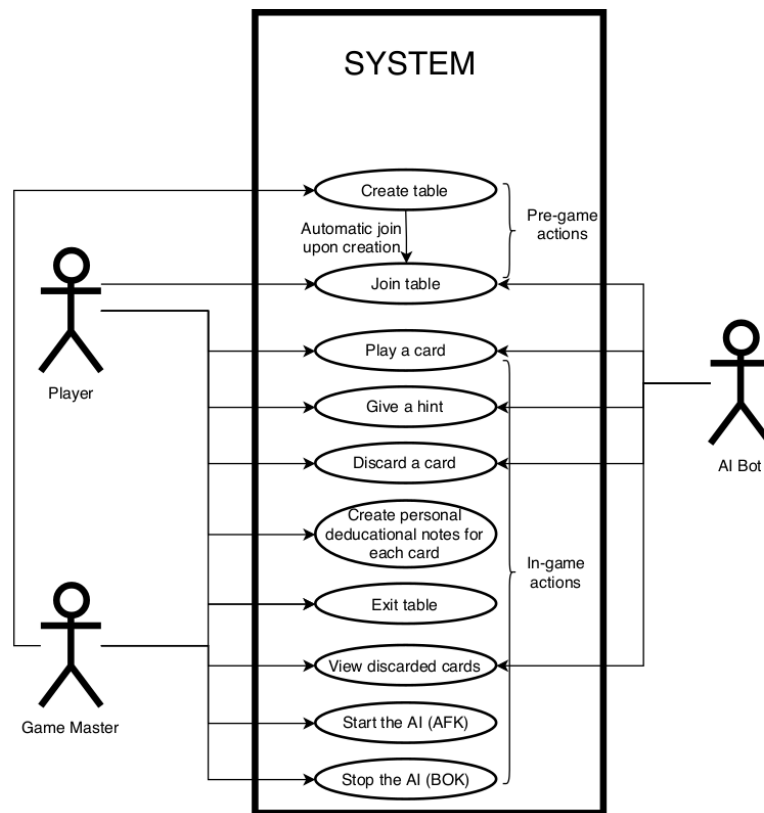
AFK At any point during the game a player can activate the AFK (Away From Keyboard) option, which invokes an AI bot that plays for the player while they are gone.

BOK If the player has switched to AFK mode and an AI is playing for them, this function returns the control of their hand to them.

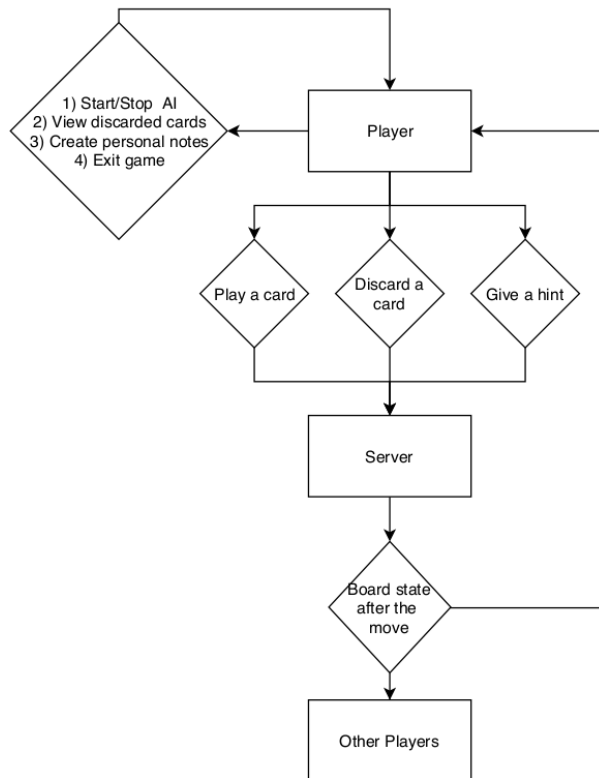
Exit Table After selecting this action the connection to the server is severed and the player is returned to the main screen.

The game is terminated when all of the in-game fuses run out, there are no more cards to draw (each player is allowed one more turn after this happens) or one or more of the players disconnect from the server.

1.3 Use Case Diagram



1.4 Data Flow Diagram



This diagram outlines a round of play, and shows how the control is switched from player to player.

2 User Characteristics

It will be assumed that most users will have basic computer usage skills such as:

- General mouse usage (can click, drag-and-drop, etc.)
- General keyboard usage

- Interface knowledge (understands that things that are greyed out are unavailable)

It will also be assumed that users who wish to use the AI functionality of the software (sending an AI to play an entire game) will have the ability to use a command line.

3 Constraints

This section outlines the basic hardware and software constraints that will be required of the end user in order to run the software.

3.1 Platform

The software will be designed to run on the JVM and should run on any machine with an installation of the J2SE (Java Platform, Standard Edition). This should allow for the software to run on most standard, modern operating systems such as MacOS, Windows, and Linux.

3.2 Network Protocols

The software will use a standard network protocol to communicate with the server. This means that support for TCP will be required.

A two-way network connection is required to allow clients to ask requests and receive messages from the server. The networking protocols used for communication between clients and the server of the system should guarantee data packets being sent and received.

3.3 Timeframe

The timeframe for the project is three months.

4 Assumptions and Dependencies

This section covers the general assumptions that are made during the requirements phase, and the dependencies that will most likely arise during the development, and why they will be included in the project.

4.1 Libraries

1. **Java Swing:** This library will support general GUI design of the software. It will be necessary for basic dialogue boxes, error reporting, and window drawing.
2. **JavaFX:** This library will be used to draw the visual portion of the game
3. **EJML:** This library will be used by the AI for matrix calculations, as well as keeping track of various local game-state variables during play.
4. **Java.net:** This will be the library responsible for addressing the general networking concerns of the software.
5. **AspectJ:** This extension of Java will aid in developing the software from an aspect-oriented approach.

Part III

Specific Requirements

1 External Interfaces

After launching the application, the user will be presented with a title screen (fig. 3, Appendix A). The options given on this screen will allow users to create a new game table, join an existing game, start a tutorial, change some basic game settings, and lastly, exit the game.

1.1 User Interface

a) Title Screen Interface:

If the user chooses to create a new game table, a dialogue box for creating games will be displayed (fig. 4, Appendix A). Users need to enter name of the game table and the maximum number of players that will be allowed in this game. Finally, a create game button and a cancel button will be provided.

Next, if the player selects the 'join an existing game' option they will again be shown a dialogue box. This dialogue box will prompt the user for the name (or address of the game they wish to join) and will provide the user a cancel button that closes the dialogue box.

If the user selects 'options', then the game will display a series of local options that can be modified by the player. This will allow them to slightly customize their client.

If the user selects 'tutorial', the game will show a short animation of a few rounds of play, explaining the basic rules of the game.

Lastly, if the user selects 'exit game', the client will close gracefully.

b) Game Table Interface:

After successfully creating a game or joining an existing game, the basic game interface will be displayed (fig. 5, Appendix A). The table will be separated into two main parts. The left part of the screen will be the main game table, it is the primary area of play (akin to a real table). Each (other) player will have an area on the screen where

they're identified, and where the player (client) will be able to see their cards.

Cards in the player's (client) hand will be placed directly at the bottom of the screen. At the top right corner of each card, there will be two small tag areas (fig. 6, Appendix A) which allow the player to mark some simple deductions on the back of the card. In the middle area, all cards that have been successfully played will be put in order and distinguished by colour.

The right side of the screen will be utilized as a 'game status' area. From top to bottom, all discards can be selected to view, time remaining will indicate time left in current turn, the number of cards left will also be shown. In addition, the number of hint tokens and fuses left on the table will also be displayed.

1.2 Hardware Interface

Since the application does not require any application specific hardware, this section will be brief. The basic hardware requirements are addressed by a keyboard and mouse. The network interfacing can be done by any standard, modern, network interface card.

1.3 Communications Interface

Communication between different interfaces would be handled by the underlying operating system, and the communication between clients and server would use a common network protocol.

2 Functions

2.1 Validity Checking Inputs

The system shall check the inputs from the user. At any point in the control flow of the software, where user input is required, validity checks will be done.

2.2 Sequence of Operations

This section serves to outline the basic pre and postconditions for each of the major operations.

Create Game:

Preconditions:

- The user is not in a game

Postconditions:

- A new game is created, and the user is automatically joined

Join Game:

Preconditions:

- The user is not in a game

Postconditions:

- The user is entered into a game

Exit:

Preconditions:

- None

Postconditions:

- The game client is closed

Discard:

Preconditions:

- The user is in a game
- The user has selected a card
- Less than eight information tokens are available

Postconditions:

- The selected card is added to the discard pile
- A new card is added to the players hand
- One information token is replenished

Play Card:

Preconditions:

- The user is in a game
- The user has selected a card

Postconditions:

- The selected card is played
- A new card is added to the player's hand

Give Hint:

Preconditions:

- The user has selected a card to give a hint about
- The user has selected the type of hint to give (colour or number)
- At least one information token is available

Postconditions:

- An information token is used
- The selected player is given the hint information

AFK:

Preconditions:

- The user is in a game

Postconditions:

- An AI player takes over for the player
- The user is put into the AFK state

BOK:

Preconditions:

- The user is in the AFK state

Postconditions:

- The AI relinquishes control to the user
- The user is taken out of the AFK state

Start AI:

Preconditions:

- A game exists for the AI to join

Postconditions:

- The AI plays a game from start to finish

View Discards:

Preconditions:

- The user is in a game
- At least one card has been discarded

Postconditions:

- The user is presented with a list of discarded cards

Take Notes:

Preconditions:

- The user is in a game
- The user has selected a card

Postconditions:

- A card in the users hand is marked with some information

2.3 Responses to Abnormal Situations

This section outlines the typical responses to abnormal working behaviour in the program.

2.3.1 Network Communication Errors

When any network communication fails, the client will attempt to resend a message to the server. If, after a certain amount of time no response is received, the client will again attempt to send a message to the server. If after these two attempts it receives to acknowledgement, then the client will consider the connection broken and exit the game (returning to the main screen).

2.3.2 Error Handling and Recovery

We will consider three primary types of errors:

- Insignificant Errors: These errors are known bugs that should not affect the functionality of the game in a major way. These will include things like:
 - Minor graphical issues
 - Minor display issues
 - Typos or grammatical errors
- Minor Errors: These errors are known bugs that do not affect the functionality of the software in a major way, but potentially make it more inconvenient to use. This will include bugs like:
 - Mislabelled icons
 - Graphical issues which obfuscate information
- Major Errors: These errors are both known, and unknown bugs which will either cause the software to stop working entirely, or stop working as intended. If a known bug in this category is encountered during runtime, the software will report an error code and attempt to exit gracefully. This category includes errors such as:
 - Any standard Java runtime error such as Array Index Out-of-bounds
 - Any complete networking failure
 - Graphical issues which cause the program to be impossible to use

3 Performance Requirements

3.1 Number of Users

The client will support a direct connection between two users. A server, and an end user. The server itself will be responsible for handling multiple users, but from the perspective of any individual client running on a machine, there are only ever two users connected to it at any given time.

3.2 Information Handling

The client will be responsible for communicating with the server, and receiving communications from the server. The parsing of information from the server format will need to be fast enough to create a smooth experience for the end user. Although "smooth experience" is hard to quantify, it should still be verifiable in the testing phase.

When the client is running in AI mode, it will need to be able to calculate its best move and send a properly formatted packet in the given time-out timeframe. This means that the basic functionality of the AI must be reasonably efficient given its limited time resources.

4 Adaptability Requirements

Any change in the requirements of the project will start a process in which the change is incorporated by editing all documentation starting with the requirements, all the way to the current project state. Changes should be subject to the same rigour as any feature that was introduced in the initial staging of the project, and be brought into the working feature set with the same process as any other feature.

Part IV

Verification Requirements

This part will outline the various verification requirements that will be imposed on the functional parts of the software. It will describe quantifiable verification methods for all operations that have failure states, as well as methods to gauge progress in the harder to quantify aspects of the program such as GUI, usability, maintainability, and adaptability.

1.1 User Interface

The verification of the user interface would be done by walking through every part of the game. For each different screen or dialogue box, every button or selection option that is provided will be tested in several conditions. This will allow verification that all graphical components are working correctly

1.2 Hardware Interface

A hardware that just meet the minimal requirements will be provided to run the game application. If the application runs well on this hardware, it will be considered as appropriate to run at any hardware that would have better factors. +

1.3 Communications Interface

The application will be tested on different operating system that all able to use common network protocol such as TCP/IP.

2 Functions

2.1 Validity Checking Inputs

The verification of correctness for validity checking will be done by attempting a range of (known) invalid inputs for every functional component that takes user input. The test cases for a functional component will be designed around trying to capture as many edge-cases as possible. The function will

be considered verified if it ignores all invalid input in the testing suite. The general set of cases to be considered will be highlighted in parallel to the sequence of operations section.

2.2 Sequences of Operations

This section will highlight the type of input validation that will be done for various operations. Note that any function which does not have any invalid input cases (like exiting the game) are not included.

Create Game:

Invalid Inputs:

- Connect to a non-existent server
- Overflow the text input field
- Make a game with too many players (more than 5)
- Make a game with too few players (1)

Join Game:

Invalid Inputs:

- Connect to a non-existent server
- Connect to a non-existent game
- Overflow the text input field
- Join a game which is full

Discard:

Invalid Inputs:

- Attempt to discard a card when the token pile is full
- Attempt to discard a card which the player does not hold

Play Card:

Invalid Inputs:

- Attempt to play a card which the play does not hold

Give Hint:

Invalid Inputs:

- Attempt to give a hint when no information tokens are available
- Attempt to give an invalid hint (both colour and number)

2.3 Responses to Abnormal Situations

2.3.1 Overflow and Underflow

To verify that overflow and underflow type situations are handled properly, any function which deals with numerical values will be put through a testing suite consisting of various overflow/underflow situations. If a function behaves robustly in all cases, it will be considered verified with respect to this section.

2.3.2 Network Communication Errors

Verification of correct behaviour during network communication errors will be done by confirming attempts to reconnect (resend) network information within a given time frame. Once the software is able to detect an error and respond accordingly within a specified real-time timeframe, it will be considered verified.

2.3.3 Error Handling and Recovery

Finally, confirming that error handling and recovery happen in an appropriate manner can only be done on a case-by-case basis. When an error is discovered, it will initially be 'patched' in an attempt to handle and isolate the error, this will be categorized (severity) and outlined (nature). The bug will now be considered "known", and appropriate recovery code will be called when this bug is triggered. The appropriate recovery response will be relative to the severity of the error. Confirming that this system is correct will be done by creating test cases consisting of "fake bugs", that trigger the desired response.

3 Performance Requirements

This section aims to outline the verification process for the performance aspect of the software.

3.1 Information Handling

To verify the performance aspect of the networking function, a series of tests that aim to quantify the network latency will be performed. The goal will be to get the client behaving in a way that most users would agree is "smooth". Although this is hard to quantify directly, a series of user-experience based questions should give adequate information on if we're heading in the correct direction. Some examples of questions that could aid in quantifying the performance of the game are:

1. Did the game feel more, or less responsive?
2. Were you frustrated with the loading times?
3. Where can the experience made to be more enjoyable (from the perspective of performance)
4. Did you notice any considerable slowdown at any point?

After every build that addresses the issues of performance, the team will discuss how the new build felt, and if the alterations to the client side performance were an enhancement or a detriment to the enjoyability of using the application.

Another tool that could be looked at to help determine if performance requirements have been met are examining the FPS, the network response time, and the profile of the software while running, but these are only tools to help guide the process, as verification will be considered complete when a unanimous decision is reached within the team.

When running in AI mode, it will suffice to generate tests that will quantify response time, and use this as a metric to verify that the AI can, within some level of certainty, respond to the server with a valid move in a given period of time.

4 Maintainability Verification

A maintainability index will be calculated using both lines-of-code and number-of-modules as a pair of quantifiable metrics which will attempt to predict the amount of effort that will be required to maintain the program. The goal will not be to explicitly use this index as a measure of success but as a rough heuristic to help guide development.

The application of this metric should help to maximize the useful life, efficiency, and safety of the software, while leaving some space for future improvements. Keeping the project simple should facilitate easier bug fixes in shorter amounts of time.

5 Reliability Verification

A simple reliability index will be used to quantify the readiness of the system. First, a set of quantifiable metrics will be used to guide the process of reliability engineering: failure rate, known remaining bugs, and testing coverage. Secondly, the individual issues which these categories address will be weighted based on the severity of the bug, ranging from serious (the bug causes an unexpected fatal error) to trivial (the bug causes a small graphical error for a moment).

This scheme aims to guide the design and testing of a reliable system and giving a valuable metric to measure progress and quality with. However, like the maintainability index, the goal is not to maximize this value, but to use it as an aid to help guide the engineering process.

Part V

Appendices

6 Appendix A: Diagrams and Figures

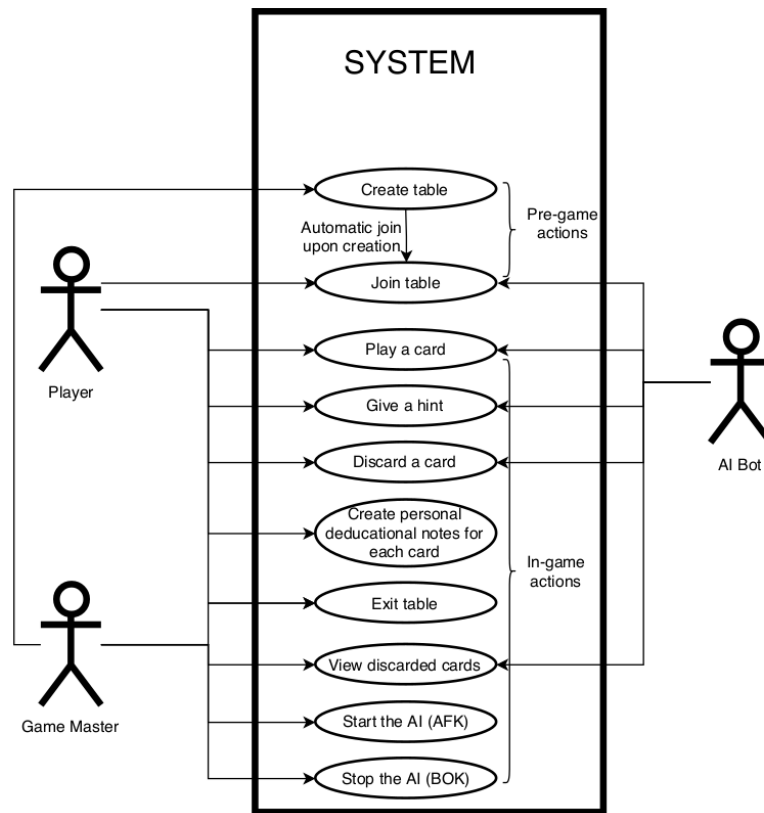


Figure 1, Use-Case Diagram

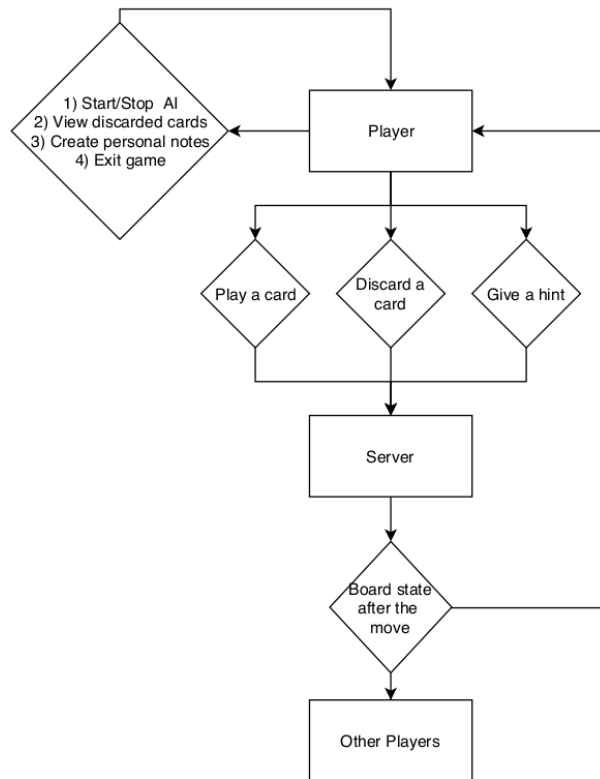


Figure 2, Data Flow Diagram

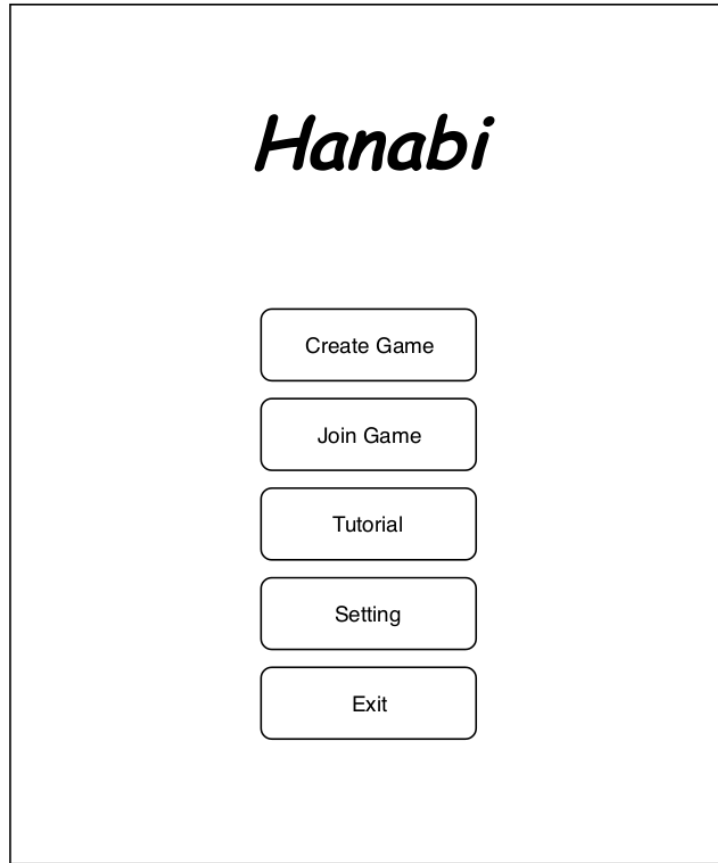
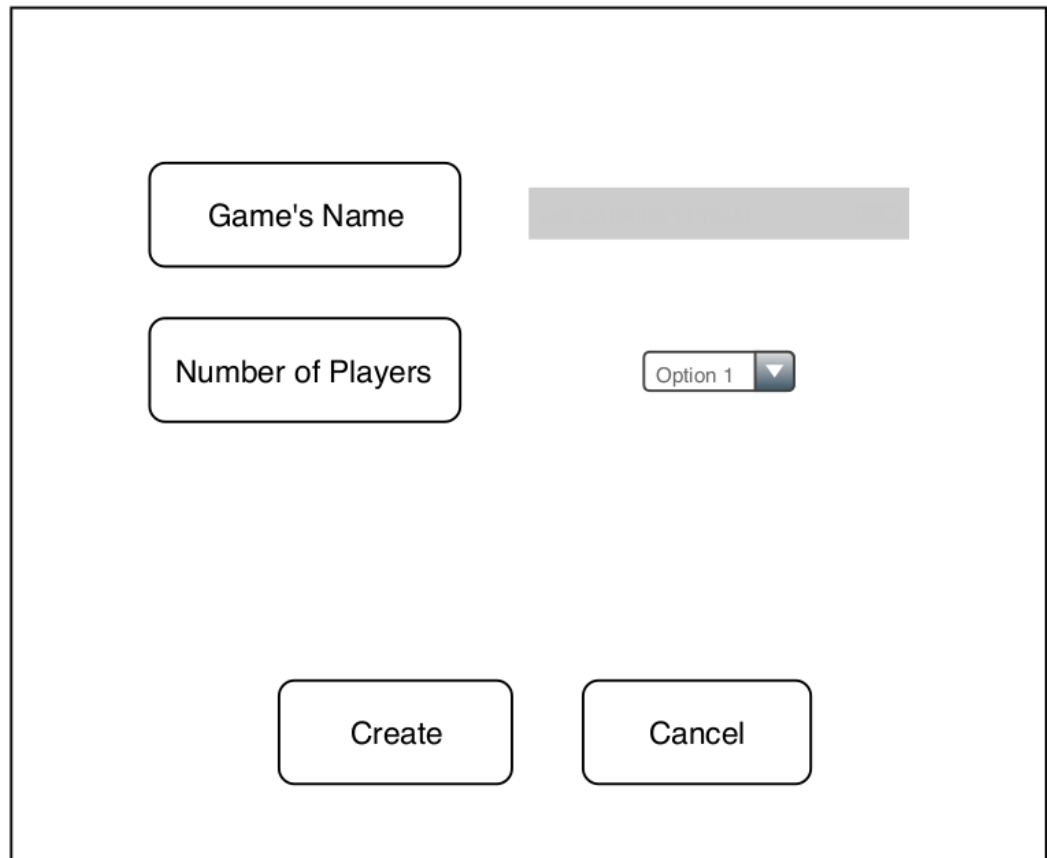


Figure 3, Title Screen



A "Create Game Dialogue" form with a light gray background and a black border. It contains two input fields on the left: "Game's Name" and "Number of Players". To the right of "Game's Name" is a gray rectangular input area. To the right of "Number of Players" is a dropdown menu showing "Option 1" with a downward arrow. At the bottom are two buttons: "Create" and "Cancel".

Game's Name	
Number of Players	Option 1 ▼
Create	Cancel

Figure 4, Create Game Dialogue

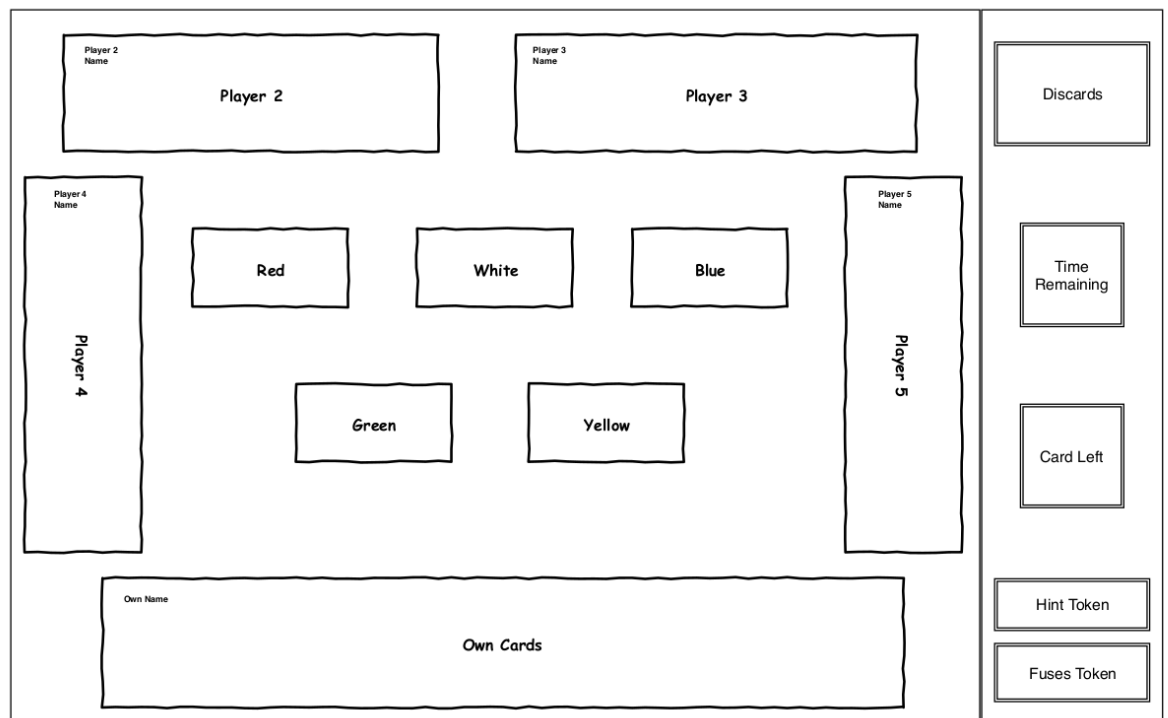


Figure 5, Game Table

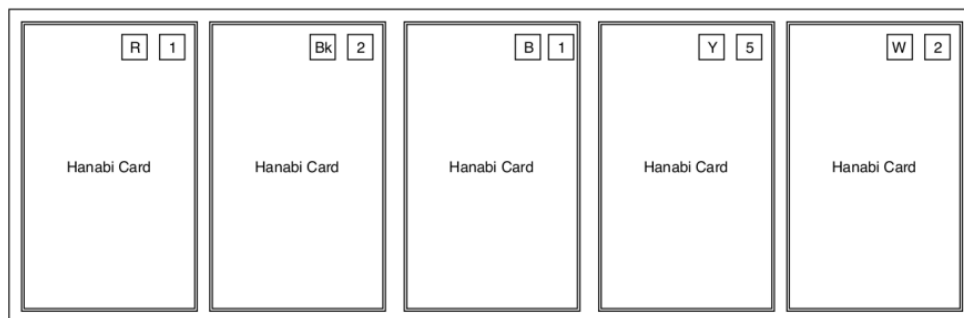


Figure 6, Player Hand

7 Appendix B: Comparison between this Document and the ISO standards

This appendix will outline the differences between the ISO standardization for the requirements documentation, and this documentation. The sections that were not included from the ISO were taken out because they didn't apply to the software as a whole, or because they were too specific for business application. Some extra sections were added that are not part of the SRS standard, these sections were added to help aid in the verification stage of the project, giving the team a general idea for verification requirements.

	SRS Standard	Hanabi Req.
Part I Introduction	✓	✓
–1. Purpose	✓	✓
–2. Scope	✓	✓
–3. Definitions	✓	✓
–4. References	✓	✓
–5. Overview	✓	✓
Part II Overall Description	✓	✓
–1. Product Perspective	✓	
–2. Product Functions	✓	✓
–3. User Characteristics	✓	✓
–4. Constraints	✓	✓
–5. Assumptions and Dependencies	✓	✓
–6. Apportioning of Requirements	✓	
Part III Specific Requirements	✓	✓
–1. External Interfaces	✓	✓
–2. Functions	✓	✓
–3. Performance Requirements	✓	✓
–4. Logical Database Requirements	✓	✓
–5. Design Constraints	✓	✓
–6. Software System Attributes	✓	✓
–6.1. Reliability	✓	✓
–6.2. Availability	✓	
–6.3. Security	✓	
–6.4. Maintainability	✓	✓
–6.5. Portability	✓	
–6.6. Adaptability		✓
–7. Organizing the Specific Requirements	✓	
–8. Additional Comments	✓	
Part IV Verification Requirements		✓
–1. External Interfaces		✓
–2. Functions		✓
–3. Performance Requirements		✓
Part V Supporting Information	✓	✓
–1. Table of Contents	✓	✓
–2. Appendices	✓	✓