The **Green** Team:
Design & Documentation
Abbreviated

Michael Philippone Todd Wright Matt SeGall
Erich Schudt

December 17, 2008

Contents

Part I
Informal Scenarios

Scenario Name: Preload
Current State: Idle server, program has not been launched, idle players, programs not launched.
Scenario: The program on the server is started, which begins the game. Players can launch their programs and connect to the server.
Next Scenario: Start Game

Scenario Name: Start game
Current State: The board is drawn and cards are shuffled to all six players. Three cards are placed at the center of the board. Miss. Scarlet (first player to join) goes first then game play moves in the order in which the other 5 players connected.
Next Scenario: Standard turn

Scenario Name: Standard turn
Current State: Player 6s turn to play
Scenario: Player 6 rolls a 3 and moves the piece 3 spaces along the board in the shape of an L and lands on a different spot on the board. Turn ends for player 6, next players turn.
Next Scenario: Standard turn, enter room, correct guess, incorrect guess, move by secret passageway, accuse, game end.

Scenario Name: move by secrete passageway
Current State: Player 5 is in a room
Scenario: Player 5 uses the secret passageway in the room and advances to the connecting room across the board. Player may guess or end turn.
Next Scenario: correct guess, incorrect guess, accuse, standard turn, game end.

Scenario Name: Enter room
Current State: Player 2 rolls a 3
Scenario: Player 2 advances 2 spaces, into a room and forfeits the last move. Player 2 may guess or end turn.
Next Scenario: correct guess, incorrect guess, accuse, move by secrete passageway, game end, standard turn.

Scenario Name: Incorrect Guess
Current State: Player 3 just entered a room
Scenario: Player 3 is in a room and guesses that Mr. Green committed the murder with a wrench in the currently located room. Mr. Green is moved from his location to the room player 3 is currently situated. Player 1 shows player 3 the wrench card and player 3s guess is disproved. Player 3 ends turn.
Next Scenario: standard turn, correct guess, incorrect guess, move by secrete passageway, accuse, game end.

Scenario Name: Correct Guess
Current State: Player 3 just entered a room
Scenario: Player 3 is in a room and guesses that Miss Scarlet committed the murder with a wrench in the room currently located at. Miss Scarlet is moved from her location to the room player 3 is currently situated. No player had the cards mentioned in player 3s guess. Player 3 ends turn.
Next Scenario: standard turn, correct guess, incorrect guess, move by secrete passageway, accuse, game end.

Scenario Name: Accuse
Current State: Player 4s turn
Scenario: Player 4 makes an accusation. Player 4 accuses Mrs. White of committing the murder with a knife in the bathroom. Player 4 looks at the center cards and does not find the knife card. Player 4 cannot move or guess or accuse again. Player remains only to disprove other guesses. Next players turn.
Next Scenario: standard turn, correct guess, incorrect guess, move by secrete passageway, accuse, game end.

Scenario Name: Game end
Current State: Player 2s turn
Scenario: Player 2 makes an accusation. Player 2 accuses Mrs. White of committing the murder with a wrench in the kitchen. Player 2 looks at the center cards and finds the exact same cards just mentioned. The cards are revealed to all players and the game is over, player 4 wins.
Next Scenario: NA

Part II
Supported Activities

1. Game Begins (System Start)
   (a) Allow players to connect and assign first connected player Miss Scarlet
   (b) 'shuffle' cards, choose the three mystery cards and deal the remaining
2. Typical Turn (no room entered)
   (a) System will automatically rolls for the player
   (b) System will determine all possible locations to move the players piece based on the dice roll
   (c) Player chooses to which location they would like to move their piece
3. Typical Turn (room entered)
   (a) System will automatically rolls for the player
   (b) System will determine all possible locations to move the players piece based on the dice roll
   (c) If player can and chooses to, allow player to enter a room
   (d) Game should dissolve remaining number of spaces left to move in turn
   (e) Allow player option to make a guess
      i. Player can name a suspected player, a weapon and the currently occupied room as a suggestion for 'Whodunnit?'
   (f) Moving in play order, the first player with at least one of the three guessed cards will be pointed out
      i. The pointed-out player can choose which cards (if possible) to show
   (g) During a guess, the suspected characters avatar is moved to the suspected room, they are allowed a guess on the start of their next turn.
4. Typical Turn (Secret Passage)

(a) Player in a room will be allowed the choice to move by spaces or (if possible) by Secret Passage

(b) Player can move by secret passage instead of by spaces and will move to the opposite corner room from the current room by the use of the secret passage

   i. Upon entering new room, player can choose to make a guess

5. Guess
   (a) Whenever a player enters a room on their turn, or is in a room because they were dragged their on a previous guess, they may make a guess
   (b) Player must guess which 2 cards (player and weapon, room the guessing player is in is implied) they think may be in the mystery folder.
   (c) The first player, going in the order of turns, who can disprove the guess must show one of their cards that was one of the cards guessed.
   (d) A correct guess will result in no one disproving the guess, and gameplay continues.
6. Accusation
   (a) At any time and in any room a player can opt to make an accusation as to 'Whodunnit?', with what and where
   (b) Player then is allowed to see the mystery cards
      i. If they are right, they win, if not, they lose and are only allowed to disprove other players suggestions / guesses
7. Win (system terminate)
   (a) If a player makes a correct accusation, they are the winner and the game ends

   OR

   (b) Last player with a piece in play wins by default
   (c) After a winner is determined, the system will halt execution and disconnect all players

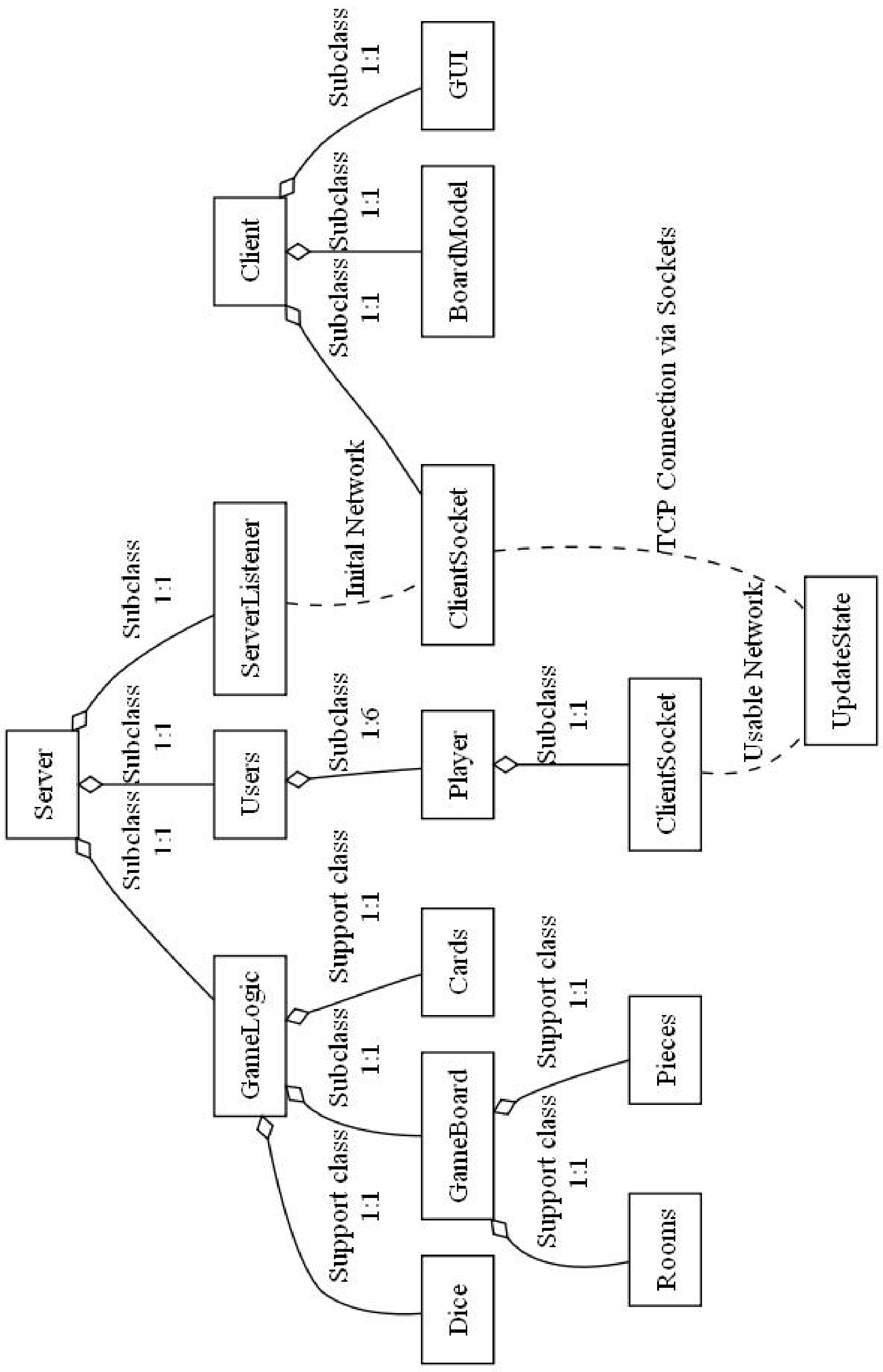
Part III
Primary Classes

Class List

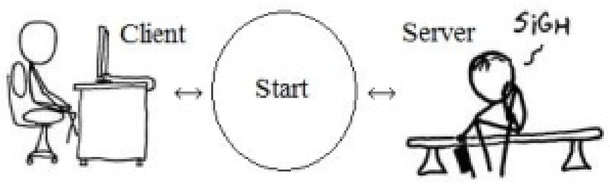
- Server
  ? *represents the entire servers driver code*
  - GameLogic
    ? *Maintain the game state ? Keeps track of whose turn it is, etc.*
    - Dice
      ? *Provides a means of determining spaces to move (simulates Die or spinner)*
      ? *Only contains a method that returns a random number between 1-6*
    - GameBoard
      ? *Maintains and manages logical model of game board*
      - Rooms
        ? *An array of (rooms, passage, board-location) tuples ? An array of non-spaces tuples*
      - Pieces
        ? *An array of (player, location) tuples*
    - Cards
      ? *An array to hold the weapon, room, character cards*
  - Users
    ? *Keep track of who is connected, and what their state is (connected/playing, connected/non-playing, etc)*
    - ServerListener
      ? *Handles initial connection requests from the clients*
      ? *All communications after initial are handled by ClientListener in Player class*
    - Player
      ? *Maintains and manages server-side instance(s) of a connected player ? Threaded*
      - ClientSocket
        ? *Handles socket communications with each player*

- Client
  ? *Represents the entire client-side driver code*
  - BoardModel
    ? *Maintains and manages model of game board to render in Gui*
  - ClientSocket
    ? *maintains socket connection with server*
  - GUI
    ? *Maintains and manages Graphical display*
    - {vast collection of support classes}
      ? *The buttons, labels, containers, eventHandlers, etc.*

Class Diagram



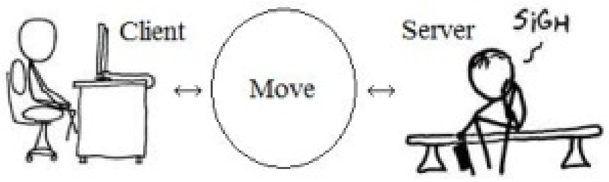
Part IV
Use Cases



Name: Start
Actors: Client, Server
Precondition: None
Trigger: Some one desires to start the game
Main Flow:

1. The server is started and program launched
2. The clients are started and programs launched
3. Clients attempt to connect to server
4. Server makes connections, remembering order that they were made
5. Server assigns pieces based on connection order
6. Server creates a board
7. Server randomizes cards, keeps a room, weapon, and person card
8. Server distributes other cards
9. Server sends board to clients
10. Server sends message to first connection beginning game play

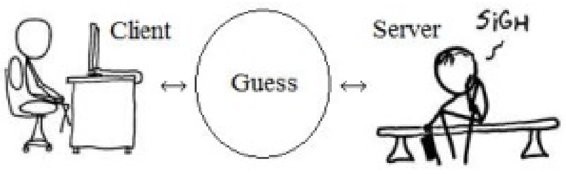
Exceptional Flow: Network error or server error
Alternate Flow: Error, not enough players joined
Post-condition: Game play in progress
Author/Date: Green Team, September 29, 2008; revised October 6, 2008



Name: Move
Actors: Server and Client
Precondition: Game play in progress, players turn
Trigger: Previous move over message received by server
Main Flow:

1. Sever generates a random integer between 1 and 6
2. Server computes all (out of a room, into a room, through passage, or along corridor) possible ending spaces from players current position
3. Server sends client updated board
4. Client chooses end position
5. Client sends position to server
6. Server updates game board
7. Server updates board for all players

Exceptional Flow: Network error
Alternate Flow: None
Post-condition: Turn continues (guess may be possible) (accuse may be possible)
Author/Date: Green Team, September 29, 2008; revised October 6, 2008



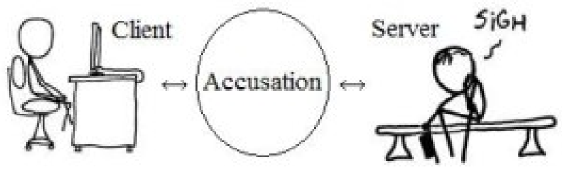
Name: Guess
Actors: Server and Client
Precondition: Players turn and located in a room that they havent guessed while being in on this turn
Trigger: Player selects guess button
Main Flow:

1. A form appears over the client GUI containing radio buttons for each player, room, and weapon
2. Player selects a combination of one weapon, one player, and the current room is pre-selected and unchangeable
3. Combination is sent to server
4. Server moves guessed players piece to guessed room
5. Server looks up player cards for first match in order of joining
6. A player that has a match is queried to show the match or one if they have multiple
7. That card is shown to guessing player along with which player had it
8. All players notified of guess made, what the guess was and that it was disproved
9. Updated board is sent to all players

Exceptional Flow: Network error
Alternate Flow: Server is unable to find a match among other players

- Message returned to guessing player that no match was found
- All players notified that a guess was made, what the guess was, and that it couldnt be disproved

Post-condition: Game state is updated to reflect changes
Author/Date: Green Team September 29, 2008; revised October 6, 2008



Name: Accusation
Actors: Server and Client
Precondition: Players turn
Trigger: Player selects accuse button
Main Flow:

1. A form appears over the client GUI containing a radio buttons for each player, room, and weapon
2. Player selects a combination of one weapon, one player, and one room
3. Combination is sent to server
4. Server verifies against saved cards
5. Server updates game board for all player with game over message, player name has won
6. Server closes connections to players
7. Server resets itself, waits for connections

Exceptional Flow: Network error
Alternate Flow:

Server finds the combo to not match.

- Players status is changed to show cards only status.
- Server sends message to players that this player made a wrong accusation, what it was, and that the player is out

Server finds the combo to not match and this is the last active player

- Server sends a message to all players telling them off
- Server closes connections to players
- Server resets itself

Post-condition: Game ends or next players turn, server get mad and game ends
Author/Date: Green Team, September 29, 2008; revised October 6, 2008

Part V
Formal Scenarios

Note: server == server machine, not a Server class object.

Startup (success)
Previous Scenario: None
Scenario:
The server-side computer is turned on and the server-side software is launched. On launch, the Server creates the GameLogic and Users objects. GameLogic creates the Cards object, Dice object, and the GameBoard object, which in turn creates the Pieces and Rooms. Users creates null Player objects and the ServerListener object.

Meanwhile, on the client-side computers the client-side software is launched. On launch the Client creates the GUI, BoardModel and ClientSocket objects. The client GUI presents a list of fields to be filled in, including player name and player IP address. Upon hitting a send button on the GUI, a message is sent to the now-waiting ServerListener. The ServerListener reads the information and creates a connection by calling on the Users to create a Player with a ClientSocket. The Users logs the order in which the Players connect, assigning pieces and starting-positions based on connection order.

Once the requisite number of Players has been reached, Game Logic (having randomized the room, player, and weapon cards as well as set one of each type aside for the goal) distributes the cardsvia the server-side ClientSocketto the Clients. The Server then pushes the GameBoard model to the connected Clients. The Clients then updates the GUI with the new board, card and player information. Finally, the Server orders GameLogic to start game play, which allows the first-connected client to begin game play.
Next Scenario: Move

Startup (fail)
Previous Scenario: None
Scenario:
The server-side computer is turned on and the server-side software is launched. On launch, the Server creates the GameLogic and Users objects. GameLogic creates the Cards object, Dice object, and the GameBoard object, which in turn creates the Pieces and Rooms. Users creates null Player objects and the ServerListener object.

Meanwhile, on the client-side computers the client-side software is launched.

On launch the Client creates the GUI, BoardModel and ClientSocket objects. The client GUI presents a list of fields to be filled in, including player name and player IP address. Upon hitting a send button on the GUI, a message is sent to the now-waiting ServerListener.

As the Server attempts to read the message, it is unable to establish a connection with the client. The Server ignores the attempt and continues waiting. The client GUI, upon not receiving a verification or connection, displays a message stating the connection was refused.
Next Scenario: None

<u>**Move**</u>
Previous Scenario: Begin or Move
Scenario:
The game play has started. A new move is determined by the GameLogic when it receives a message that the previous turn has ended from the previous player. Dice generates a random number between 1 and 6. The GameLogic determines all possible ending points based on the number from the roll. The GameBoard sendsto the players BoardModela game board to display with all the endpoints (ie: into a room, out of a room, along corridor, or through tunnel) highlighted for the player. The player then chooses which way and how far to move. This can be into a room, out of a room, through a corridor, or secret passage, all determined by the GameLogic. The player selects one of the endpoints. This new position is sent to the server. The GameLogic verifies the new position and sends it to Pieces via GameBoard and then updates all client boards.
Next Scenario: Move, Guess, Accuse

<u>**Guess**</u>
Previous **Scenario:** Move
Scenario:
The player has just entered a room and is prompted on his/her GUI to make a guess regarding that room. The player has the option to make the guess or pass on it. If they choose to make the guess they are prompted (via GUI) for a character, a weapon and the current room is implied. Once the player chooses the guess attributes, the guess is sent to the server. The Server contacts the GameLogic. The GameLogic finds which player was guessed and has the GameBoard changed so that the guessed player is moved to the corresponding room. The GameLogic determines that the guessed cards are held by no other player. All of the players are notified by the server that a guess has been made, what the guess was and that it was not disproved. All of the boards are updated to reflect the changes made by the guess.

Next Scenario: Move or Accuse

Guess (Disproved)
Previous Scenario: Move
Scenario:
The player has just entered a room and is prompted on his/her GUI to make a guess regarding that room. The player has the option to make the guess or pass on it. If they choose to make the guess they are prompted (via GUI) for a character, a weapon and the current room is implied. Once the player chooses the guess attributes, the guess is sent to the server. The GameLogic finds which player was guessed and changes the GameBoard so that the guessed player is moved to the corresponding room. The GameLogic takes the guess and matches the cards to players. The GameLogic has the Server contact the first player holding a card. The first card holder will be forced to show a card. If that player only has one, it is sent to the guessing player as proof and the card holder is notified they disproved the guess with their card. If that player has two or more of the guessed cards, that player will be prompted for which to show the player. The Server then notifies all players that a guess has been made, what the guess was and that it was disproved. All of the boards are updated to reflect the changes made by the guess.
Next Scenario: Move or Accuse

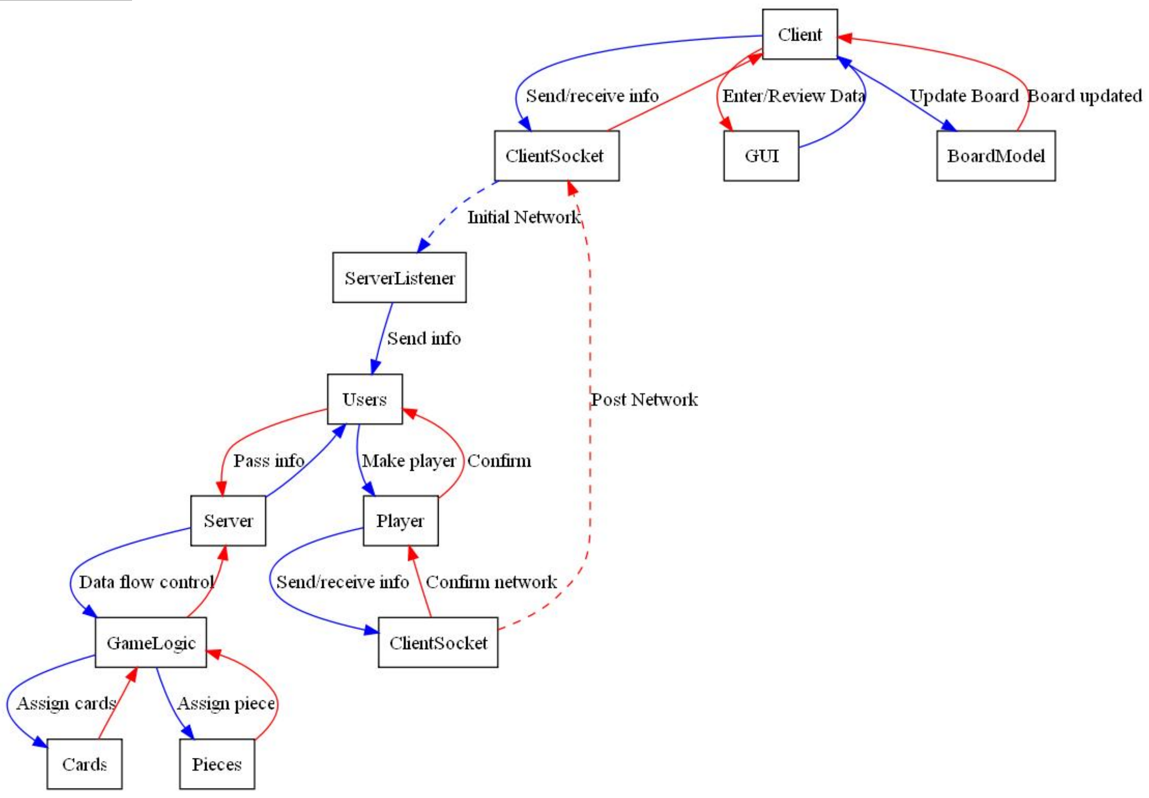
Accuse (win)
Previous Scenario: Any game-play scenario
Scenario:
The player, at any point in their turn, can elect to make an accusation and is prompted on his/her GUI to specify which character, which weapon, and which room they would like to accuse. The Server then contacts the GameLogic. It will attempt to match the accusation against the secret cards. It finds all three match. The Server then notifies all players that an accusation has been made, what the cards were and that the player has won. The Server then destroys the Players, resets itself, and waits for players to connect for another game. The Client then shows the opening screen on the GUI.
Next Scenario: NA

Accuse (lose)
Previous Scenario: Any game-play scenario
Scenario:
The player, at any point in their turn, can elect to make an accusation and is prompted on his/her GUI to specify which character, which weapon, and which room they would like to accuse. The Server then contacts the Game-

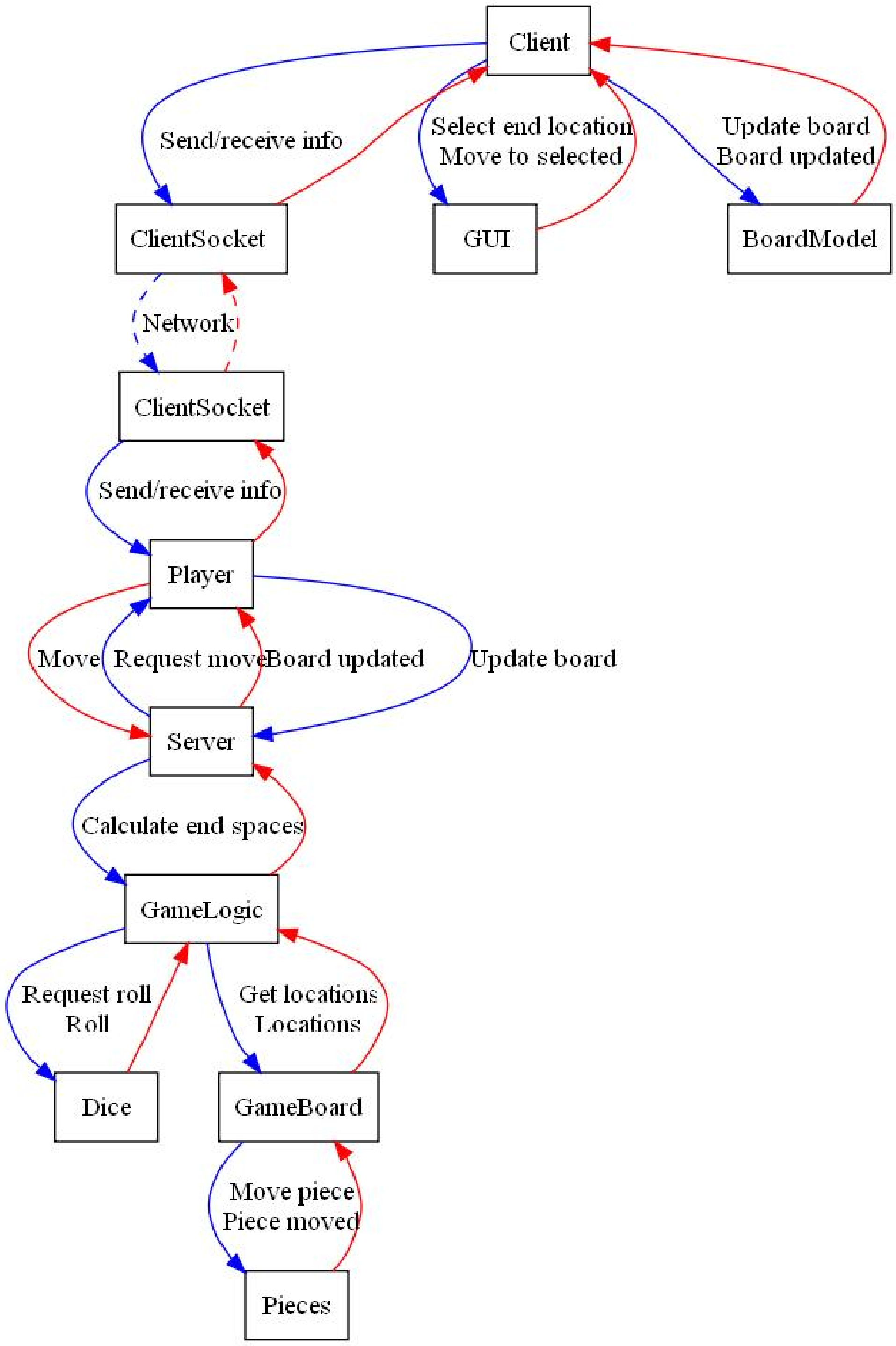
Logic which attempts to match the accusation against the secret cards. The cards do not match and the players accusation is refuted by showing him/her the appropriate secret cards. The Server then notifies all players that an accusation has been made, what the cards were and that the player has lost but will continue to play only as far as proving or disproving guesses by other players. If this player is the last player to make an incorrect accusation, the game ends with no winners and all are notified that the game is over. The Server then destroys the Players, resets itself, and waits for players to connect for another game. The Client then shows the opening screen on the GUI.
Next Scenario: NA

Part VI
Class Collaboration Diagrams

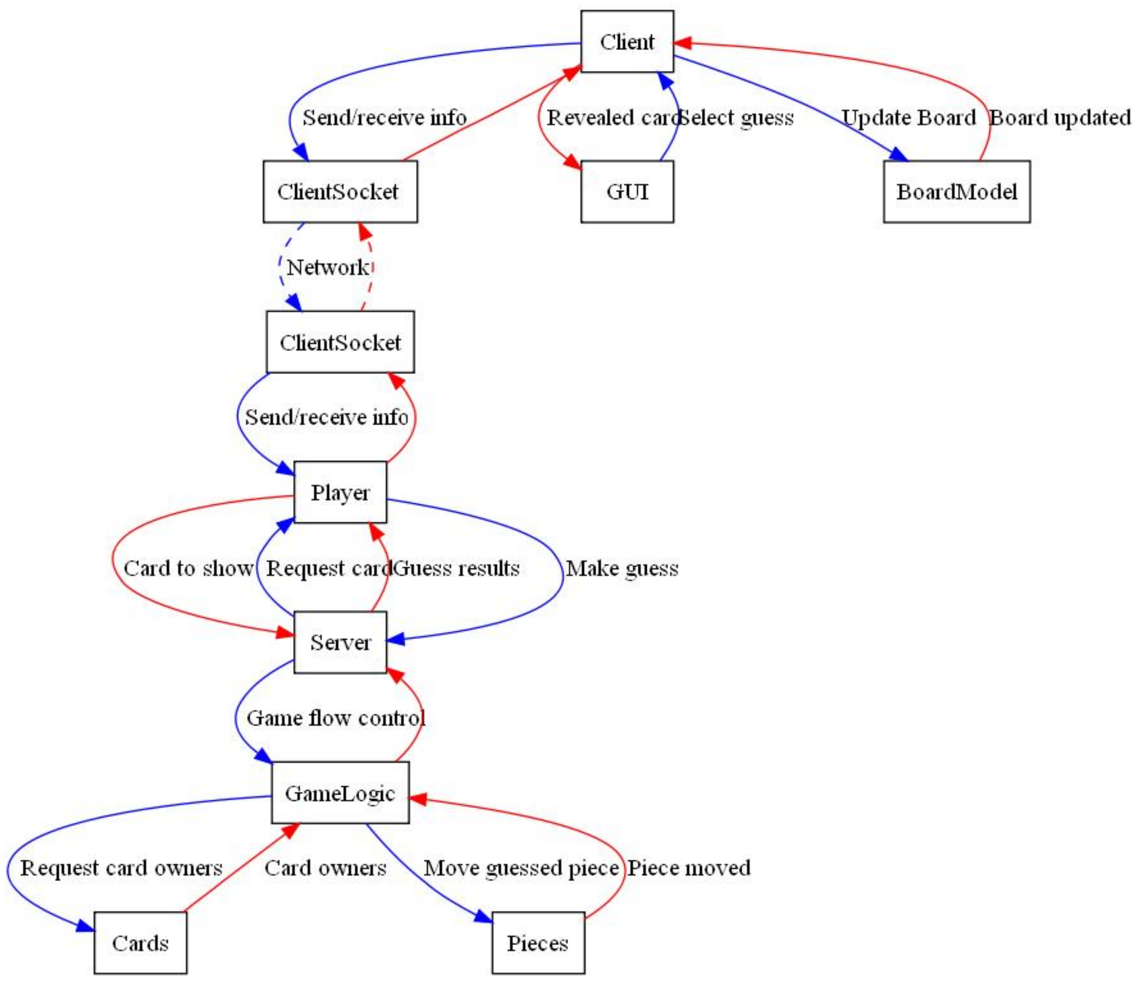
Start



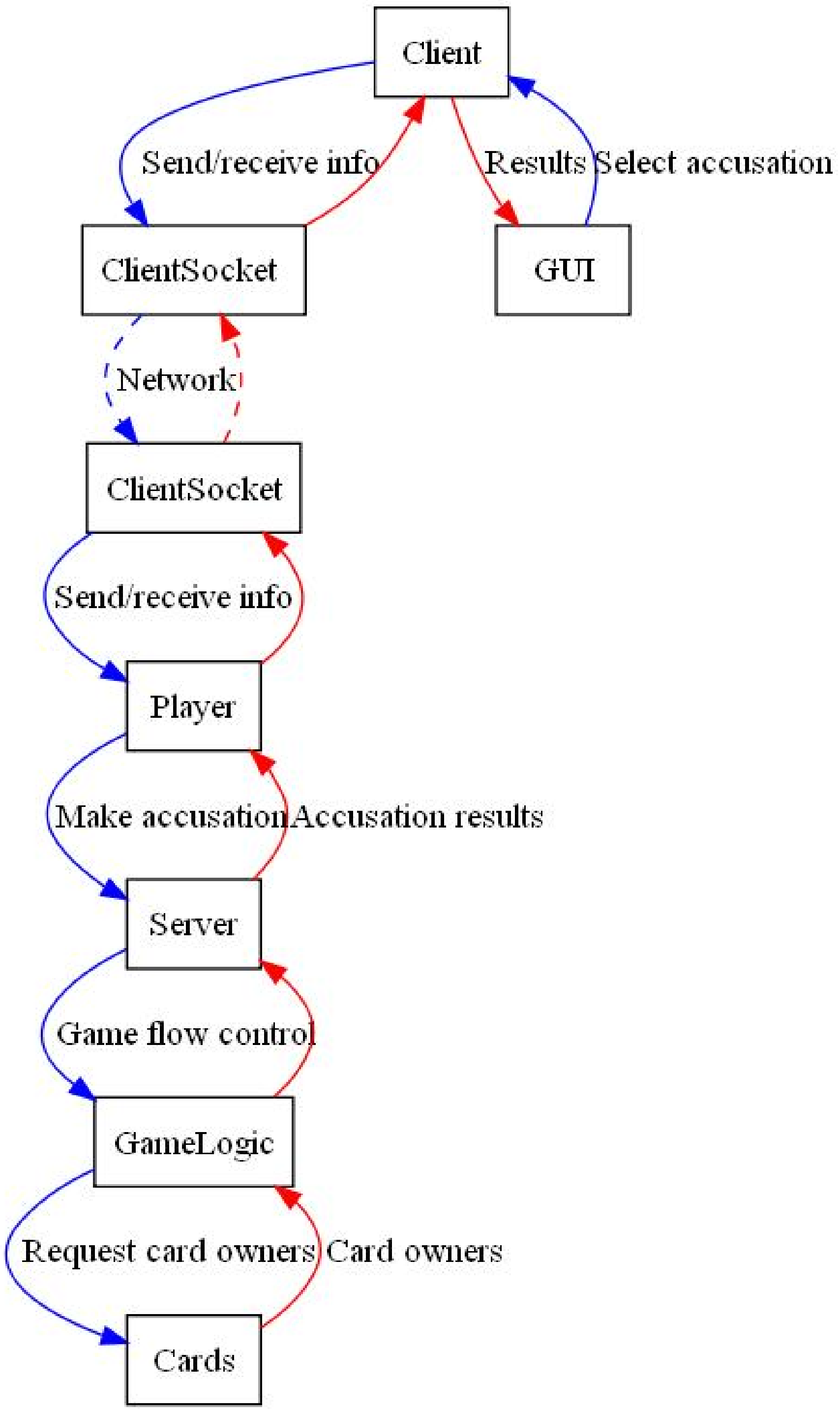
Move


Client
Send/receive info
Select end location
Move to selected
Update board
Board updated
ClientSocket
GUI
BoardModel
Network
ClientSocket
Send/receive info
Player
Move
Request move
Board updated
Update board
Server
Calculate end spaces
GameLogic
Request roll
Roll
Get locations
Locations
Dice
GameBoard
Move piece
Piece moved
Pieces

Guess

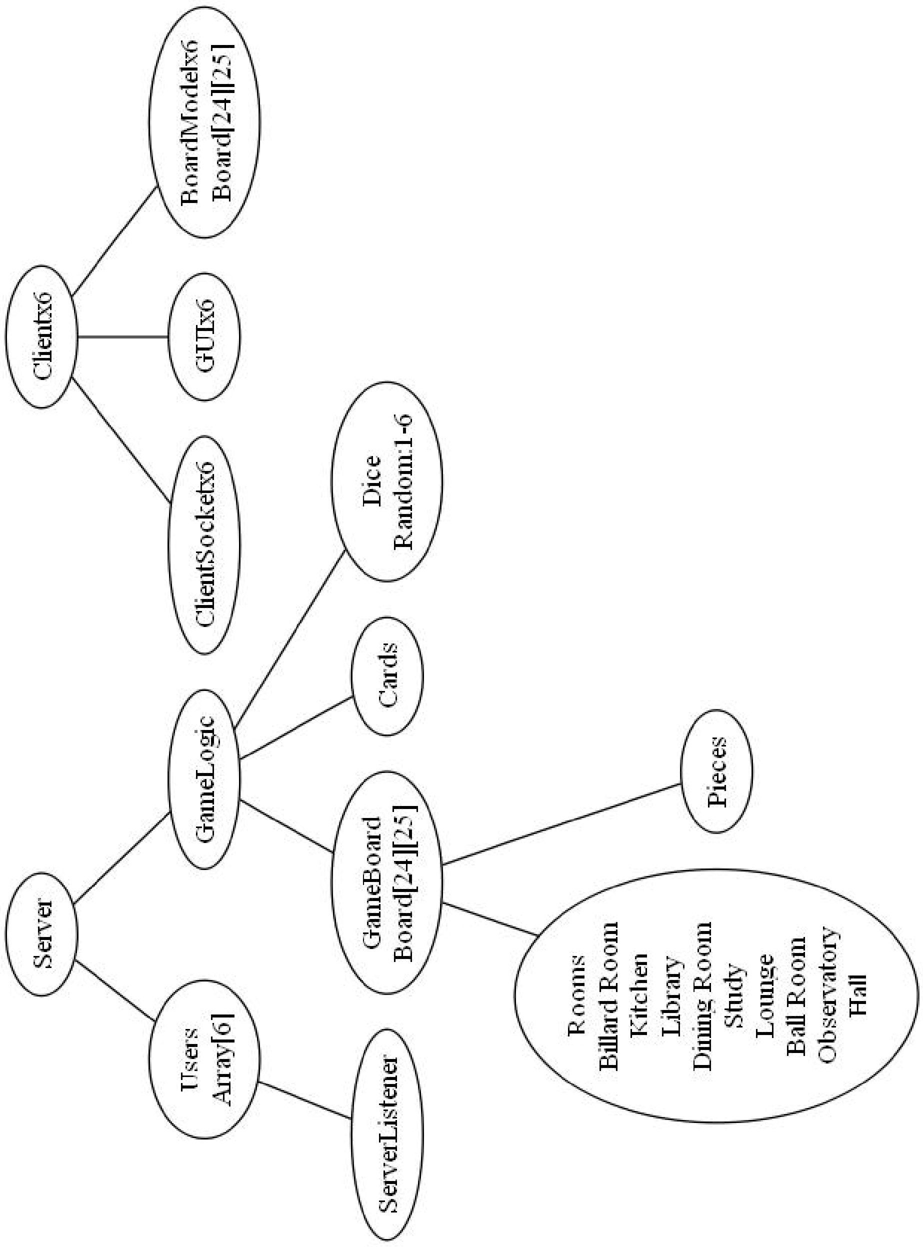

Client
Send/receive info
Revealed card
Select guess
Update Board
Board updated
ClientSocket
GUI
BoardModel
Network
ClientSocket
Send/receive info
Player
Card to show
Request card
Guess results
Make guess
Server
Game flow control
GameLogic
Request card owners
Card owners
Move guessed piece
Piece moved
Cards
Pieces

Accuse

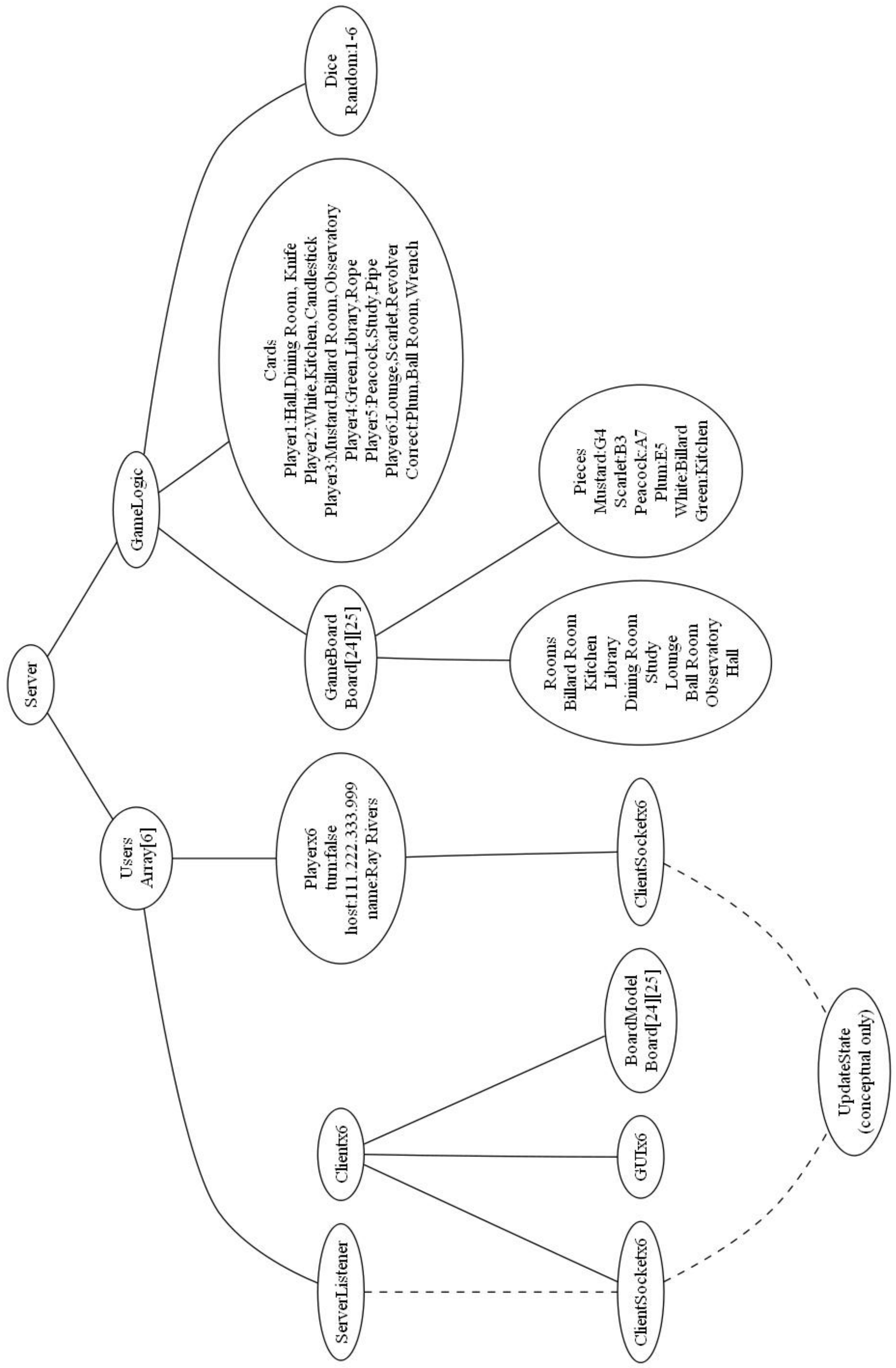

Client
Send/receive info
Results
Select accusation
ClientSocket
GUI
Network
ClientSocket
Send/receive info
Player
Make accusation
Accusation results
Server
Game flow control
GameLogic
Request card owners
Card owners
Cards

Part VII
Object Diagrams

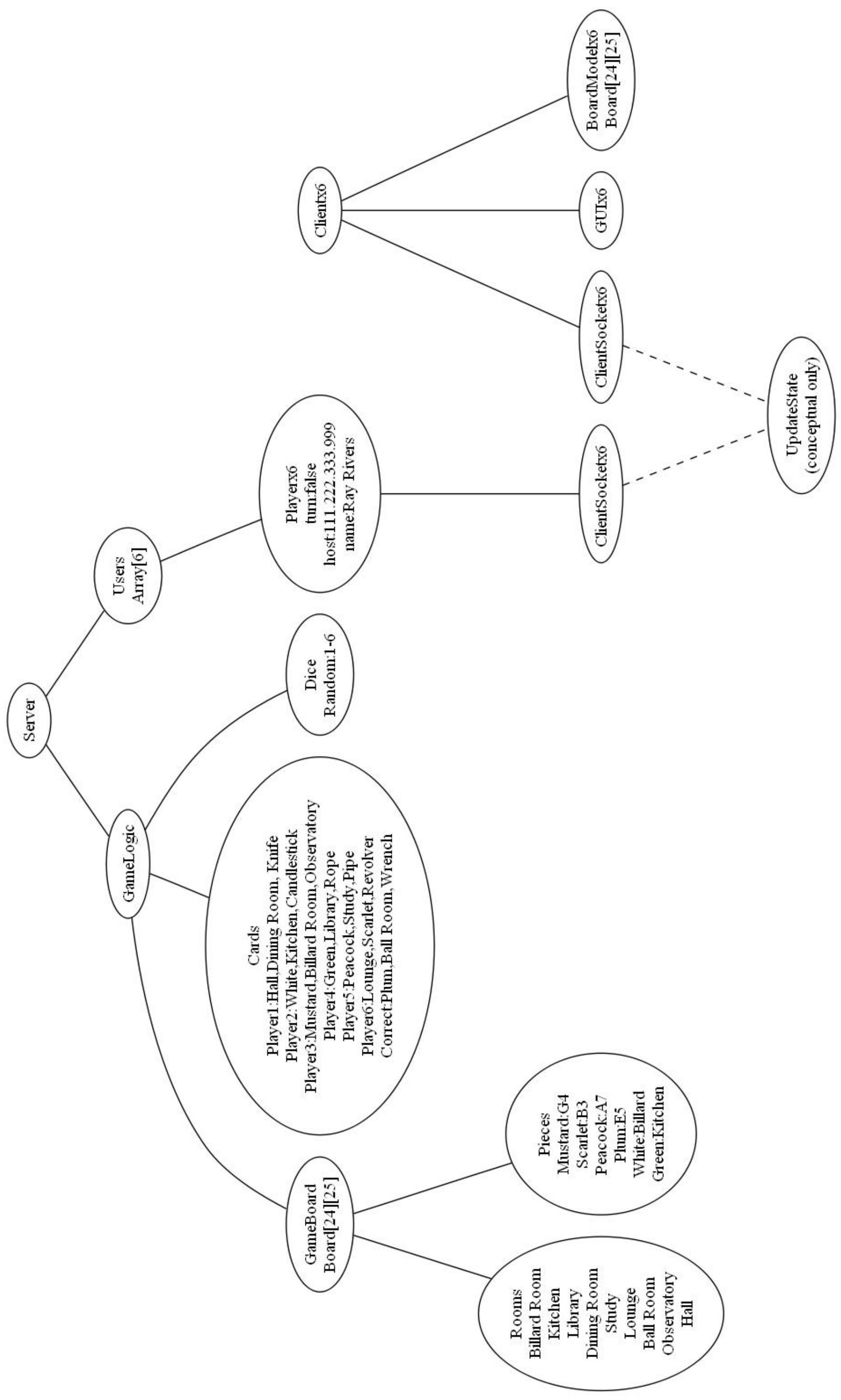
Startup



Full Model

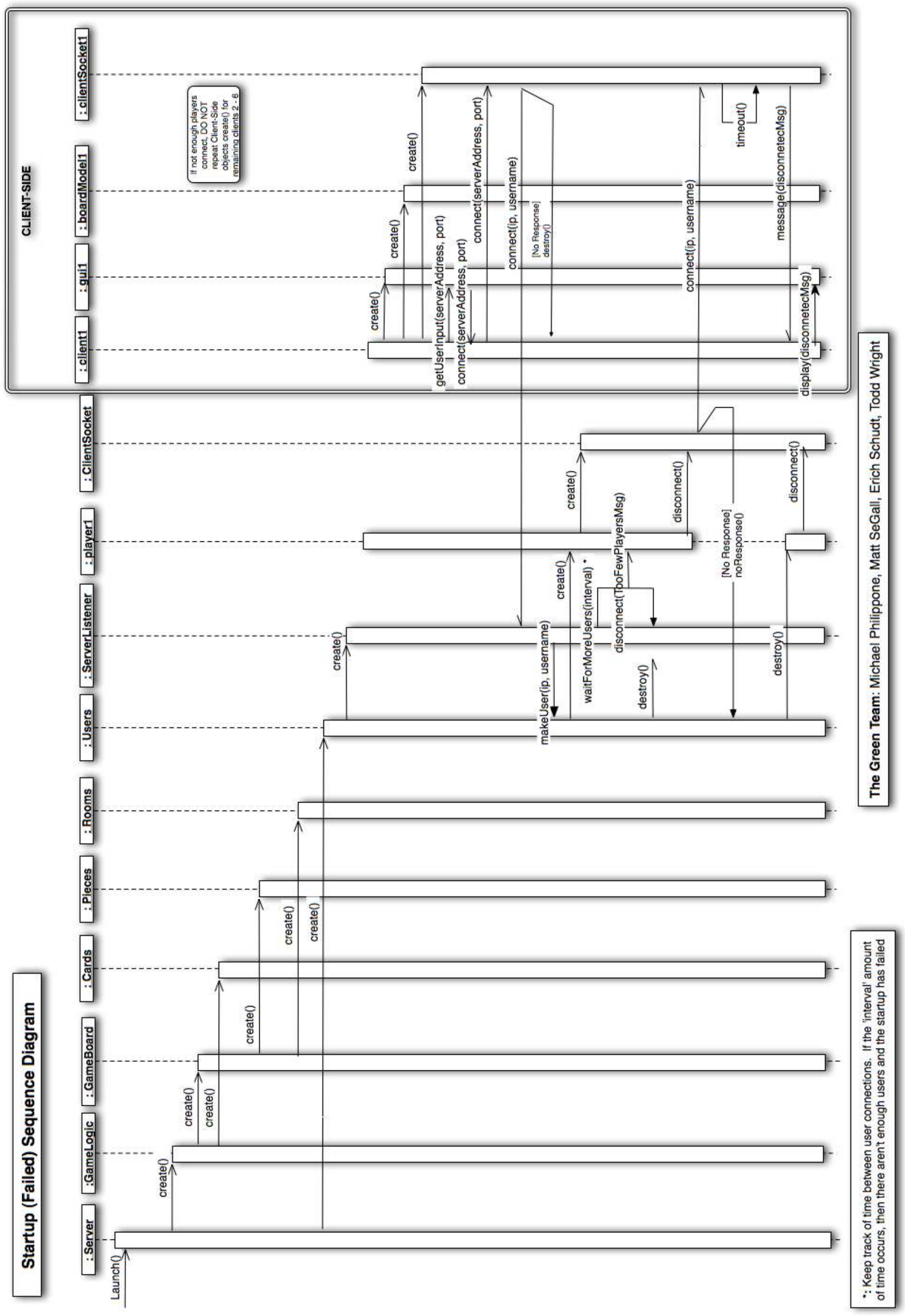

Server
GameLogic
Dice
Random:1-6
Cards
Player1:Hall,Dining Room, Knife
Player2:White,Kitchen,Candlestick
Player3:Mustard,Billard Room,Observatory
Player4:Green,Library,Rope
Player5:Peacock,Study,Pipe
Player6:Lounge,Scarlet,Revolver
Correct:Plum,Ball Room,Wrench
Pieces
Mustard:G4
Scarlet:B3
Peacock:A7
Plum:E5
White:Billard
Green:Kitchen
GameBoard
Board[24][25]
Rooms
Billard Room
Kitchen
Library
Dining Room
Study
Lounge
Ball Room
Observatory
Hall
Users
Array[6]
Playerx6
turn:false
host:111.222.333.999
name:Ray Rivers
ClientSocketx6
BoardModel
Board[24][25]
UpdateState
(conceptual only)
Clientx6
GUIx6
ServerListener
ClientSocketx6

Use *Objects used during a Move, Guess or Accusation*

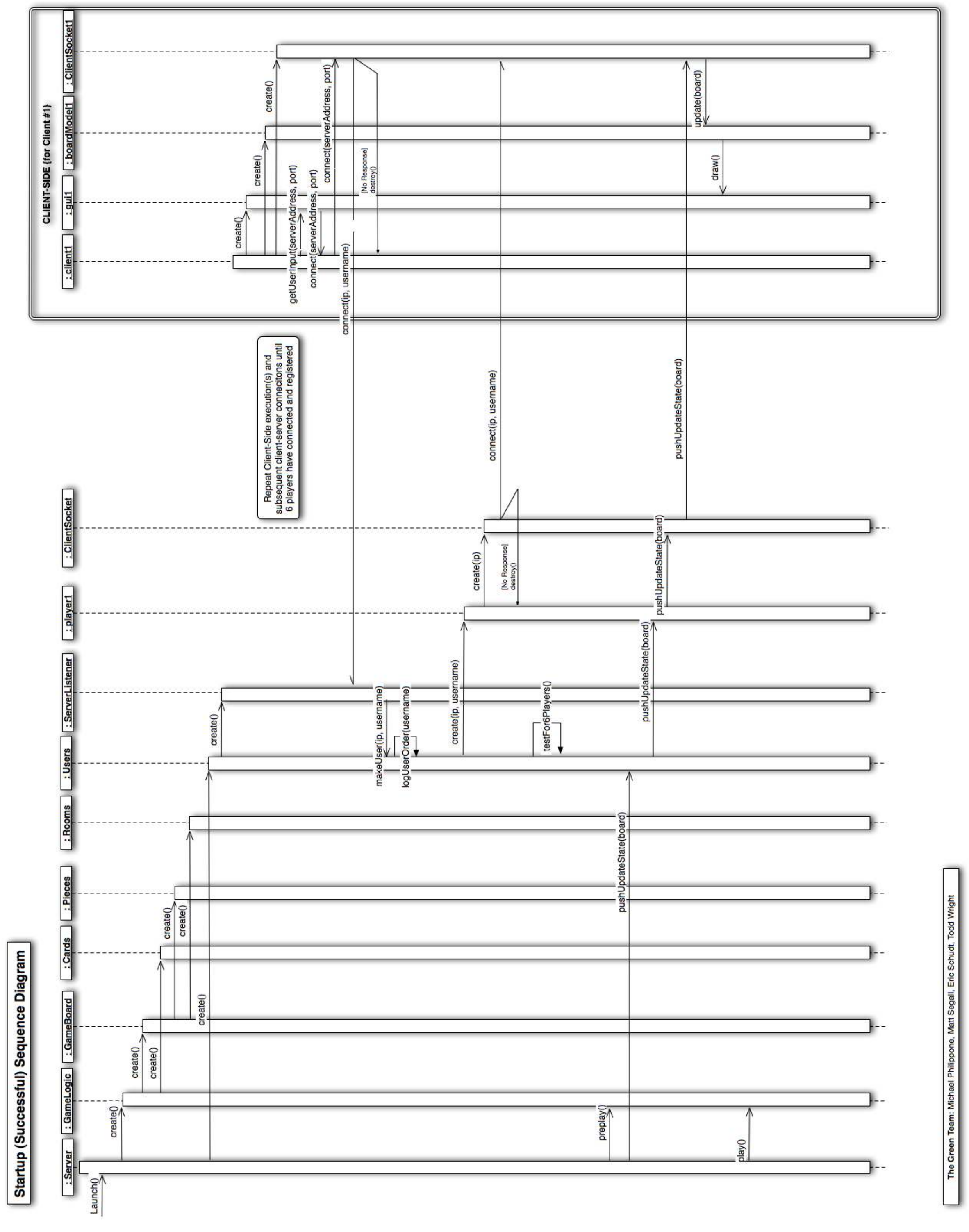

Server
Users Array[6]
Playerx6 turn:false host:111.222.333.999 name:Ray Rivers
ClientSocketx6
UpdateState (conceptual only)
ClientSocketx6
Clientx6
GUIx6
BoardModelx6 Board[24][25]
GameLogic
Dice Random:1-6
Cards
Player1:Hall,Dining Room, Knife
Player2:White,Kitchen,Candlestick
Player3:Mustard,Billard Room,Observatory
Player4:Green,Library,Rope
Player5:Peacock,Study,Pipe
Player6:Lounge,Scarlet,Revolver
Correct:Plum,Ball Room,Wrench
GameBoard Board[24][25]
Pieces
Mustard:G4
Scarlet:B3
Peacock:A7
Plum:E5
White:Billard
Green:Kitchen
Rooms
Billard Room
Kitchen
Library
Dining Room
Study
Lounge
Ball Room
Observatory
Hall

Part VIII
Sequence Diagrams

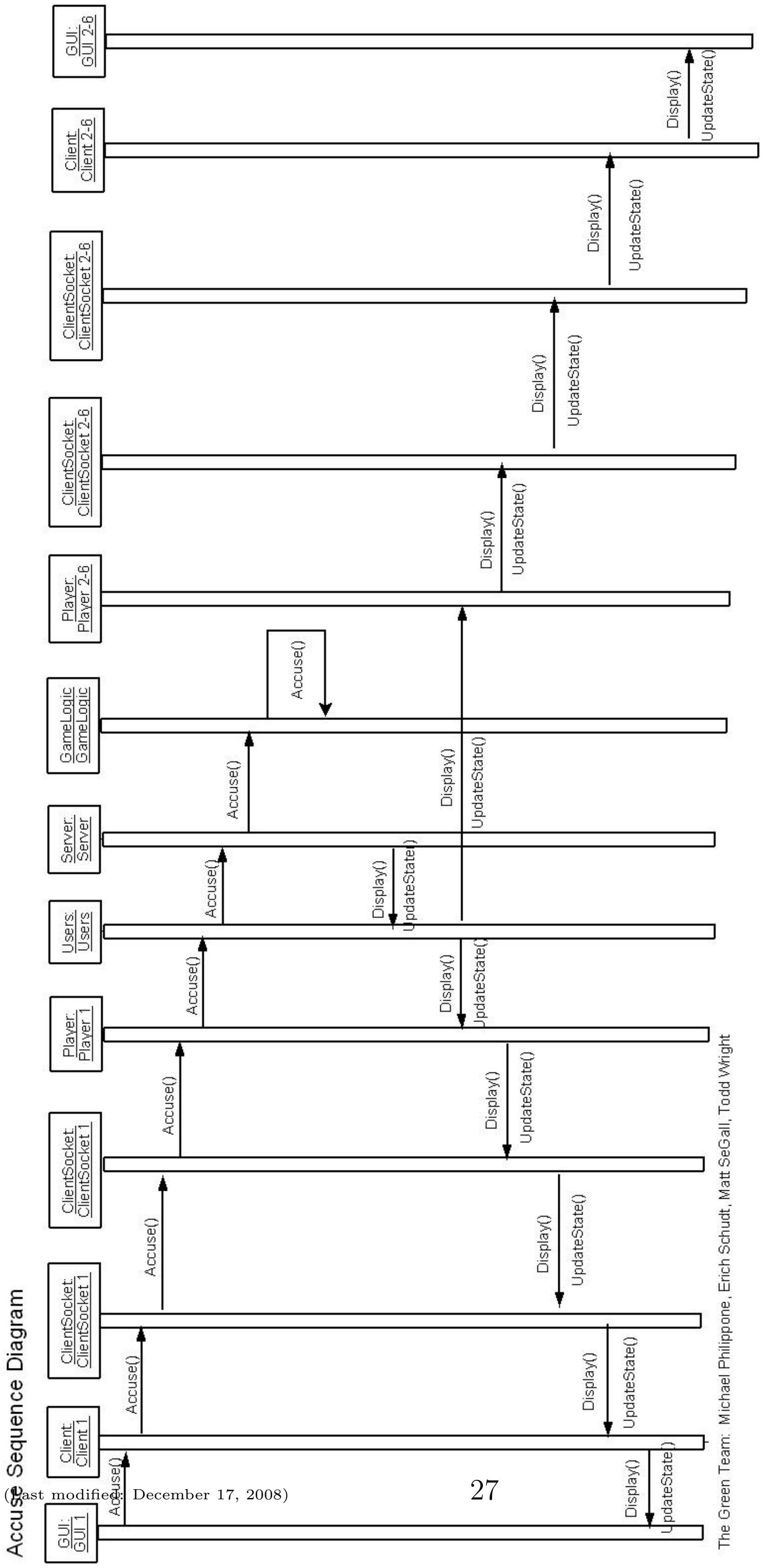
Startup (failed)


Startup (Failed) Sequence Diagram
CLIENT-SIDE
: clientSocket1
: boardModel1
: gui1
: client1
: ClientSocket
: player1
: ServerListener
: Users
: Rooms
: Pieces
: Cards
: GameBoard
:GameLogic
: Server
Launch()
create()
create()
create()
create()
create()
create()
create()
create()
create()
create()
create()
getUserInput(serverAddress, port)
connect(serverAddress, port)
connect(serverAddress, port)
connect(ip, username)
[No Response] destroy()
makeUser(ip, username)
waitForMoreUsers(interval) *
disconnect(TooFewPlayersMsg)
destroy()
disconnect()
connect(ip, username)
[No Response] noResponse()
timeout()
message(disconnetecMsg)
display(disconnetecMsg)
destroy()
disconnect()
If not enough players connect, DO NOT repeat Client-Side objects create() for remaining clients 2 - 6
The Green Team: Michael Philippone, Matt SeGall, Erich Schudt, Todd Wright
*: Keep track of time between user connections. If the 'interval' amount of time occurs, then there aren't enough users and the startup has failed

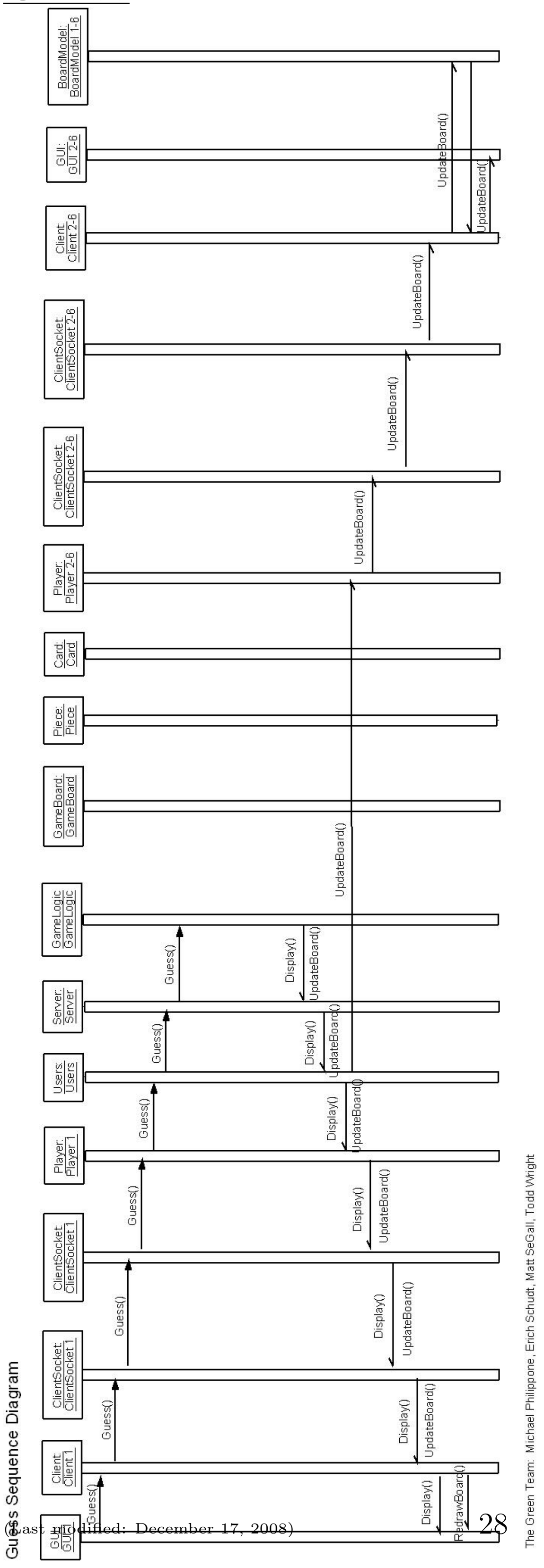
Startup (Successful)



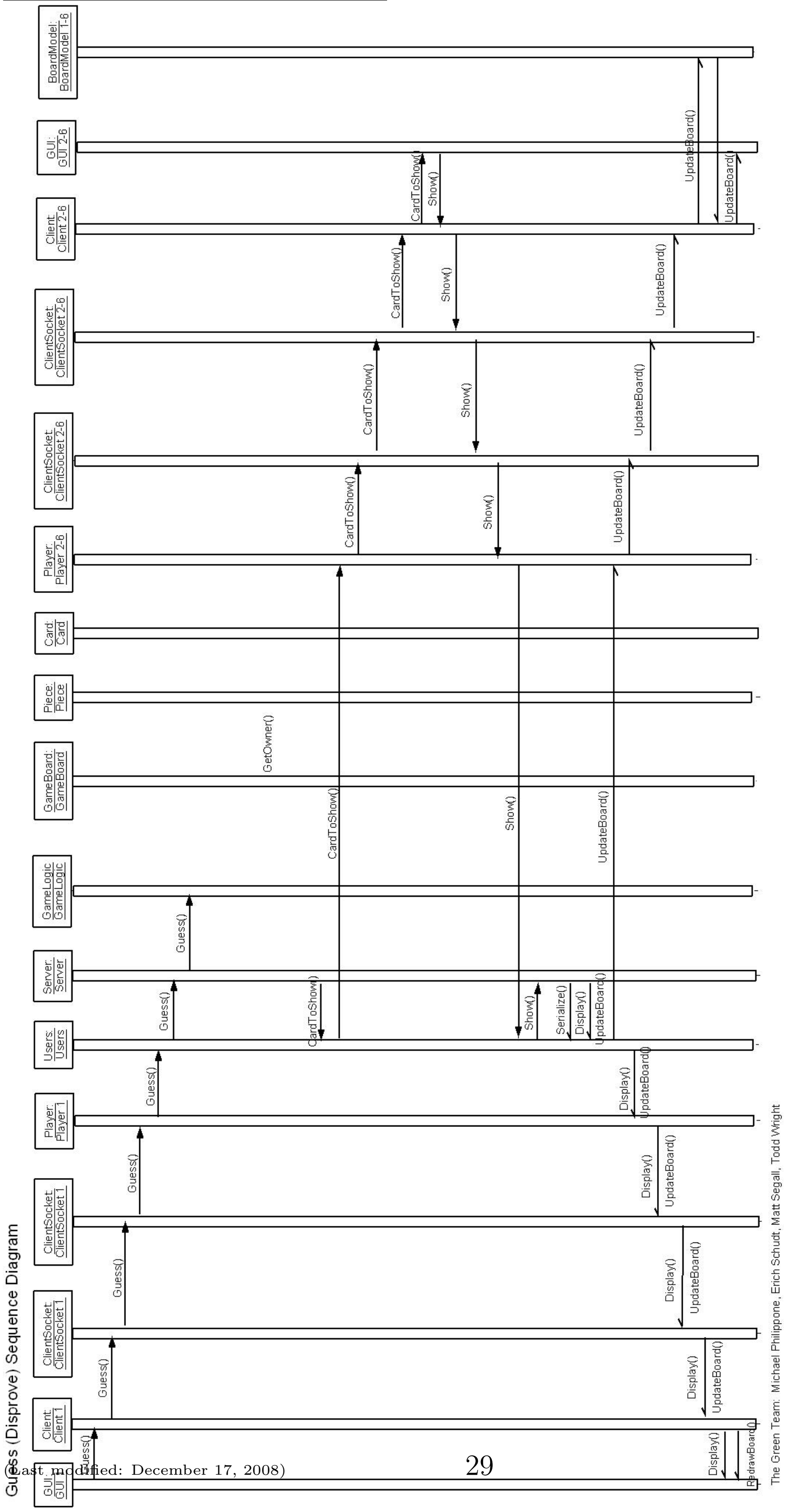
Accuse


Accuse Sequence Diagram
GUI: GUI 1
Client: Client 1
ClientSocket: ClientSocket 1
ClientSocket: ClientSocket 1
Player: Player 1
Users: Users
Server: Server
GameLogic: GameLogic
Player: Player 2-6
ClientSocket: ClientSocket 2-6
ClientSocket: ClientSocket 2-6
Client: Client 2-6
GUI: GUI 2-6
Accuse()
Accuse()
Accuse()
Accuse()
Accuse()
Accuse()
Accuse()
Accuse()
Display() UpdateState()
Display() UpdateState()
Display() UpdateState()
Display() UpdateState()
Display() UpdateState()
Display() UpdateState()
Display() UpdateState()
Display() UpdateState()
Display() UpdateState()
Display() UpdateState()
Display() UpdateState()
The Green Team: Michael Philippone, Erich Schudt, Matt SeGall, Todd Wright

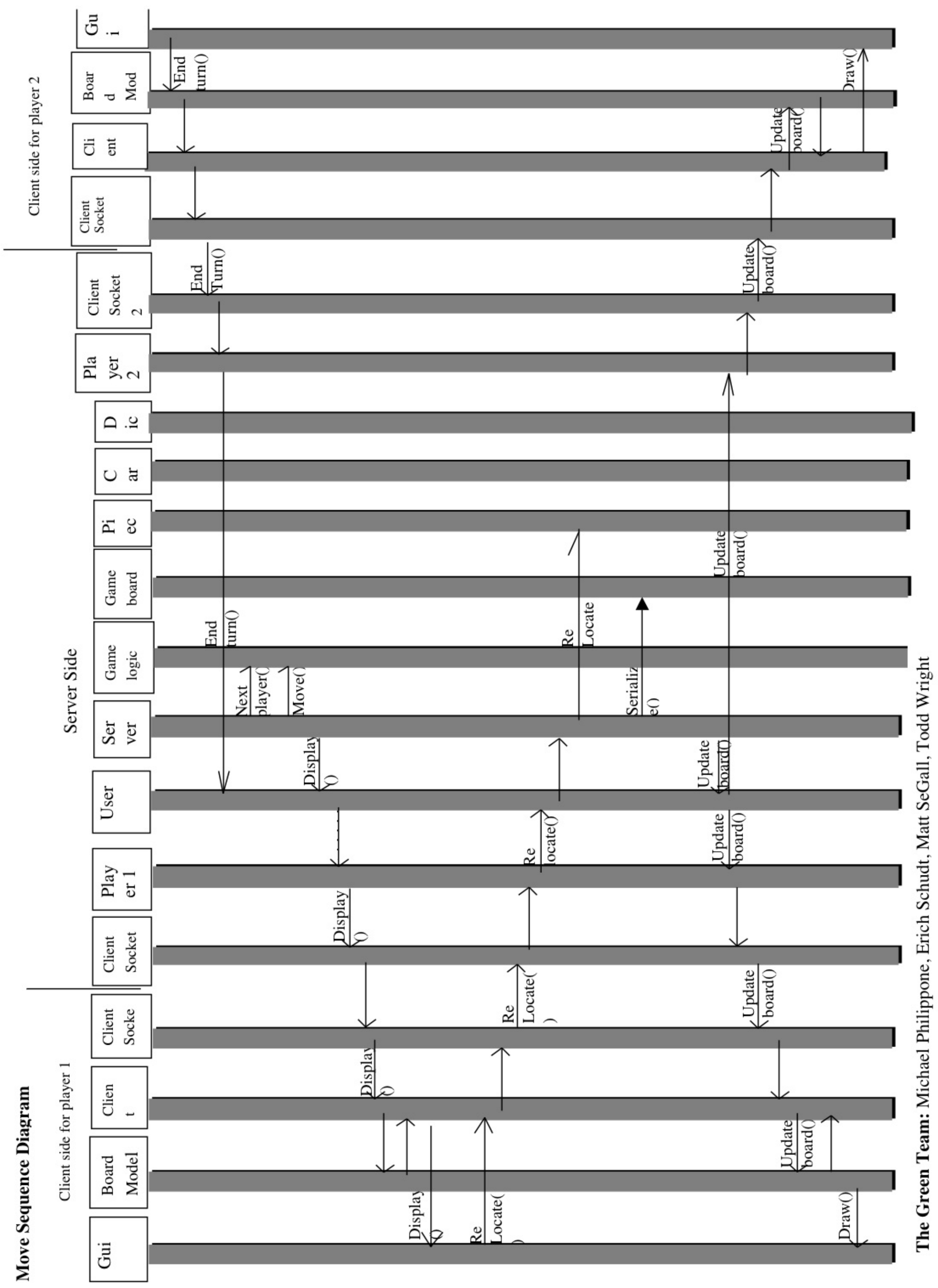
Guess


Guess Sequence Diagram
GUI: GUI
Client: Client 1
ClientSocket: ClientSocket 1
ClientSocket: ClientSocket 1
Player: Player 1
Users: Users
Server: Server
GameLogic: GameLogic
GameBoard: GameBoard
Piece: Piece
Card: Card
Player: Player 2-6
ClientSocket: ClientSocket 2-6
ClientSocket: ClientSocket 2-6
Client: Client 2-6
GUI: GUI 2-6
BoardModel: BoardModel 1-6
Guess()
Display()
UpdateBoard()
RedrawBoard()
The Green Team: Michael Philippone, Erich Schudt, Matt SeGall, Todd Wright

Guess (Disproved)



Move


Move Sequence Diagram
Client side for player 1
Server Side
Client side for player 2
Gui
Board Model
Client
Client Socke
Client Socket
Play er 1
User
Ser ver
Game logic
Game board
Pi ec
C ar
D ic
Pla yer 2
Client Socket 2
Client Socket
Cli ent
Boar d Mod
Gu i
Display()
Re Locate()
Update board()
Draw()
End turn()
Next player()
Move()
Serialize()
The Green Team: Michael Philippone, Erich Schudt, Matt SeGall, Todd Wright

Part IX
Implementation Plan

Explanation:
For our implementation we have chosen to use the Threaded design method, but with a few tweaks. We have created threads based on some obvious divisions in the system, and then within them we divided the implementation into a collection bottom-up components.

This Implementation plan was NOT adhered to. Once we began development, roles shifted and the plan morphed based on each group member's ability within the project.

- Phase 01 (9 NOV 2008): **(Server-Side Development)**
  - Network Components development (code: Mike)
  - User management components (code: Erich)
  - GameLogic Components
    - GameBoard sub-components (code: Matt)
    - GameBoard (code: Matt)
    - Dice and Cards (code: Matt)
- Phase 02 (16 NOV 2008): **(Server Unit Testing)**
  - Develop suite of drivers to test all functions
    - User creation / management tests (test: Mike)
    - Network message passing tests (test: Erich)
    - GameLogic testing (test: Todd)
      - Create a board
      - Simulate moves / guesses / accusations
      - Test for game rule adherence
      - Respond to player requests
    - Full server testing (test: Matt)
      - Listen for incoming connection requests
      - Create users
      - Communicate with users
      - Respond to and update user requests
- Phase 03 (30 NOV 2008): **(Client-Side Development)**

  - Client class (client-side driver) (code: Matt)
  - BoardModel components (code: Mike)
    - Create cell objects to represent grid squares in BoardModel grid
    - Create array to represent board
    - Code the BoardModel update functionality
  - Network components
    - from server-side development, use same socket class (code: Todd)
  - GUI
    - Board representation (code: Mike)
    - Users Cards (code: Matt)
    - Events and their listeners (code: Mike & Todd)
    - Menu Bar (code: Todd)
    - Server message display (code: Erich)
    - Game play controls (Move, Guess, Accuse, buttons) (code: Erich)

- Phase 04 (4 DEC 2008): **(Client Unit Testing)**
  - Assuming the Server is already functional:
    - Test the message passing and receiving (test: Erich)
    - Test BoardModel Updates (test: Matt)
    - Test GUIs ability to display board / game info (test: Mike)
    - Test GUIs ability to respond to user input (test: Todd)
  - Test Exceptional conditional (run-time and compile-time errors)

- Phase 05 **(7 DEC 2008)**: **(Integration Testing)**
  - Test guesses and accusations (test: Mike)
  - Test AI (test: Todd)
    - When a user quits / their connection is lost / Accuses wrongly, the computer assumes their role in playing silently

Part X
Unit Testing Plan

Server (driver class)

- ensure that server components are instantiated
- ensure that game play is looping until someone wins or all lose
  - make sure a move can be executed
  - make sure a guess can be executed
  - make sure an accusation can be executed
- catch a winning / losing condition

User Creation / Management

Users / Players

- creating 1 Player and creating 6 Players
- create 7 Players and -1 Players
- test communicating on a Players socket
- assign the player names
- verify the order of Player connections
- verify the number of Players
- player removal

Game Engine

GameLogic

- test guesses with all sorts of strings
  - (proper input(s), null input(s), anomalous input(s))
- test accusations with all sorts of strings
  - (proper input(s), null input(s), anomalous input(s))
- test movement
  - into room from hallway
  - out of a room into a hallway
  - from hallway space to hallway space
  - to room from tunnel
  - to room not using door
- make sure that nextPlayer works

Pieces

- relocating a player
  - on a move (user has chosen the space)
    - make sure chosen space is in the list of possible endpoints
  - on a guess (user has been implicated in a guess / accusation)
- all players pieces end up in their appropriate start space
  - print the board and verify the piece spaces
- test if certain cell is occupied or not
- test getting players positions

Cards

- shuffle cards
- deal the cards
- players have cards

GameBoard

- print out the board and verify against images in repository
  - REPO://src/clueServer/clueBoard.jpg
  - REPO://src/clueServer/clueBoard2.jpg
- test if all tunnel cells are such and all non-tunnel cells are such
- test if all doorway cells are such and all non-doorway cells are such
- print out the serialized version of the board and verify the pieces and rooms
- test that doorways of rooms correspond to image

Rooms

- make sure rooms are rooms and nulls (non-spaces) are nulls
  and that these correspond to the 'REPO://src/clueServer/clueBoard.jpg'

Dice

- ensure that it returns random between 1 and 6 inclusive

Network

ClientSocket

- connect to another ClientSocket instance

- read from the socket
- write to the socket
- test socket closure

ServerListener

- listen for users and receive incoming connection
- create Players in Users class

BoardModel

- make sure that a cell can be clicked
  - either makes the message for the socket or does nothing
- can be updated with a new version

GUI

- draw the board
- display the cards for a user
- get server connection info
- guesses and accusations
- moves
- ignore extraneous clicks and input
- close connections at game quit

Client (driver class)

- initiate connection with server
  - call GUI for connection info
- start users game
- play through
- handle game quit

Part XI
Socket Communication Protocol

Below is the protocol the Green Team has devised for Clue game messages on the socket

```
Protocol for Clue game Socket communication

*****************************************************
Client -> Server communications
*****************************************************

  "move <X> <Y>"
    - X: the x coordinate of the space to which to move
    - Y: the y coordinate of the space to which to move

  "endturn"
    - the player has opted to end his or her turn

  "guess <CHARACTER> <WEAPON> <ROOM>"
    - CHARACTER: the guessed character who did the crime
    - ROOM: the guessed room where the crime occurred
    - WEAPON: the weapon that was used in the crime

  "accuse <CHARACTER> <WEAPON> <ROOM>"
    - CHARACTER: the guessed character who did the crime
    - ROOM: the guessed room where the crime occurred
    - WEAPON: the weapon that was used in the crime

  "showcard <CARD>"
    - CARD: card that the user is showing to disprove a guess

  "log <STR>"
    - STR : string message to print in the Server's log

  "exit"
    - disconnect the invoking player from the game
  "testingshutdown"
     - message signifying that a partial test requiring
       Server process usage has ended and the Server should
       shut down before the game finishes

*******************************************************
Server -> Client communications
*******************************************************

  "update <X1>,<Y1>,<COLOR1>,<true/false>;...;<Xn>,<Yn><COLORn>,<true/false>"
    - X_ : an X coordinate to update on the GUI
    - Y_ : a Y coordinate to update on the GUI
    - <true/false>: the boolean logic for
      whether or not the cell at X Y is clickable
    - <COLOR> : specifies color change for cell

  "havecard <C1>;<C2>;...;<Cn>"
    - informing the Client of cards that were dealt
      (should only be used at startup)

  "choosecard <C1>;<C2>[;<C3>]"
    - C_ : card that the user can choose to show (up to three)

  "inform <STR>"
    - STR : string to show to 'inform' the person playing

  "gameover [<X>]"
    - X: the player with name X has won,
      (if specified) and the game has ended
```

Part XII
What We've Learned

"I learned ..."

Erich

- SVN: during the last week I've really gotten the hang of it
- Of course, the whole planning and designing process. I did not even know there was a universally accepted designing process or diagrams.
- The whole network programming portion. I had no clue about it before taking this class. Working on string passing has made me become quite comfortable with it.
- The importance of javadoc. It makes code easier to understand and use, especially code that someone else has written.
- Some new programs such as 'Dia', 'Graphviz' and others

Matt

- Socket communication is not as easy as it sounds
- Plan in code, not words
- There are always more conditions to check
- Doing everything in one method is not ideal for debugging
- Communication is very critical
- Code in groups (coders together in same place)
- Don't ask about other people's projects
- The internet is an invaluable resource for everything
- When it doesn't work, rewrite it from scratch a different way
- Have someone else do the whole thing

Michael

- Don't save ANYTHING until the last minute
- Set up proper group communication channels at the start of the project
  - Set up the roles for group members EXPLICITLY. Even if they change throughout, it is extrememly necessary to know who to call upon and for what.
  - Set up certain communication rules. (ie: check your email daily, 24 hours notice for a missed meeting, contact group members before and after a missed meeting / class, etc.)
- Determine everyone's responsibilities from the start of each discrete project step. No one should ever not know what he or she is responsible for. The same is true for knowing what others are responsible for.
- A solid, collaborative approach hinges on regular, documented and transparent progress
- '. . . From each according to ability . . . '

Todd

- Importance of javadoc and heavily commented code when working with a group so that even if you didn't write the code you can still understand what is happening.
- How much more complicated and complex a project becomes when you do all the paper work and planning we did.
- Importance of a group leader to keep everyone on track and up to date.
  - ("Thank you to Mike: I feel this was your role whether you agreed to it or not")
- Putting things off until later is bad!

Part XIII
Team Member Contacts

- **Name:** Michael Philippone
  Phone: 585.469.3891
  Email: `michael.philippone@gmail.com`
- **Name:** Erich Schudt
  Phone: 505.918.5413
  Email: `eschudt1@ithaca.edu`
- **Name:** Matt SeGall
  Phone: 724.813.1870
  Email: `msegall1@ithaca.edu`
- **Name:** Todd Wright
  Phone: 845.728.3256
  Email: `twright1@ithaca.edu`