The Green Team:
Design & Documentation

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

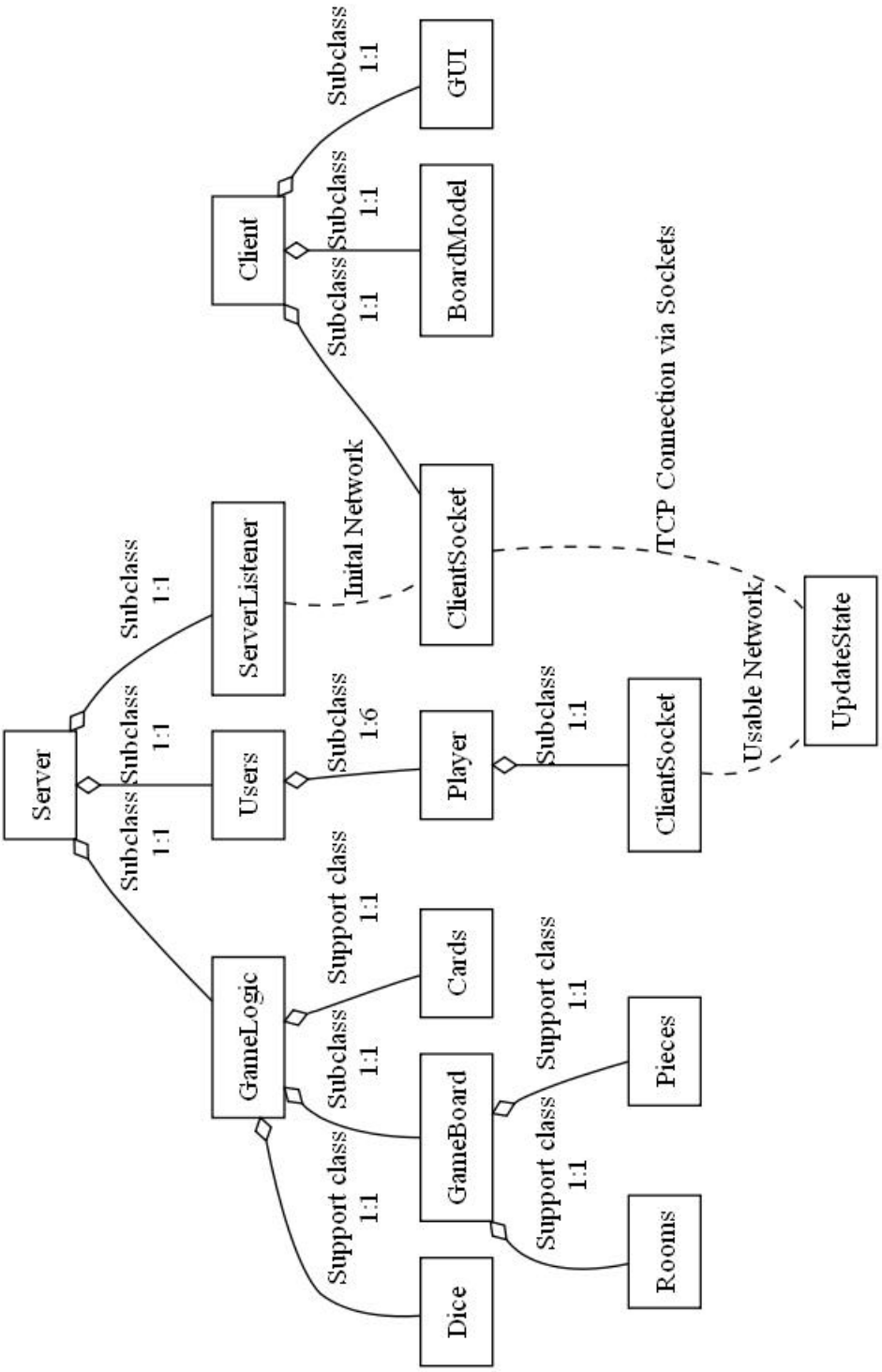
Part III
Primary Classes

Class List

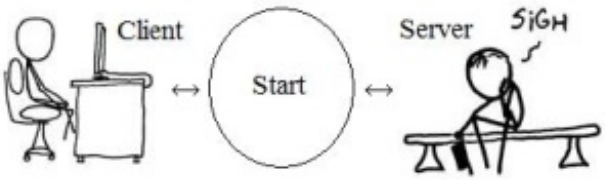
- Server
  ? *represents the entire servers driver code*
  - GameLogic
    ? *Maintain the game state ? Keeps track of whose turn it is, etc.*
    - Dice
      ? *Provides a means of determining spaces to move (simulates Die or spinner)*
      ? *Only contains a method that returns a random number between 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



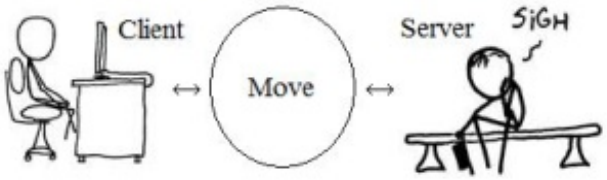
Part IV
Use Cases



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

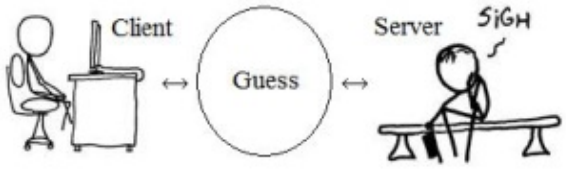
Exceptional Flow: Network error or server error
Alternate Flow: Error, not enough players joined
Post-condition: Game play in progress
Author/Date: Green Team, September 29, 2008; revised October 6, 2008



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



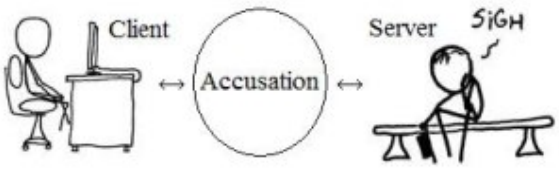
Name: Guess
Actors: Server and Client
Precondition: Players turn and located in a room that they havent guessed while being in on this turn
Trigger: Player selects guess button
Main Flow:

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

Move
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

Guess
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

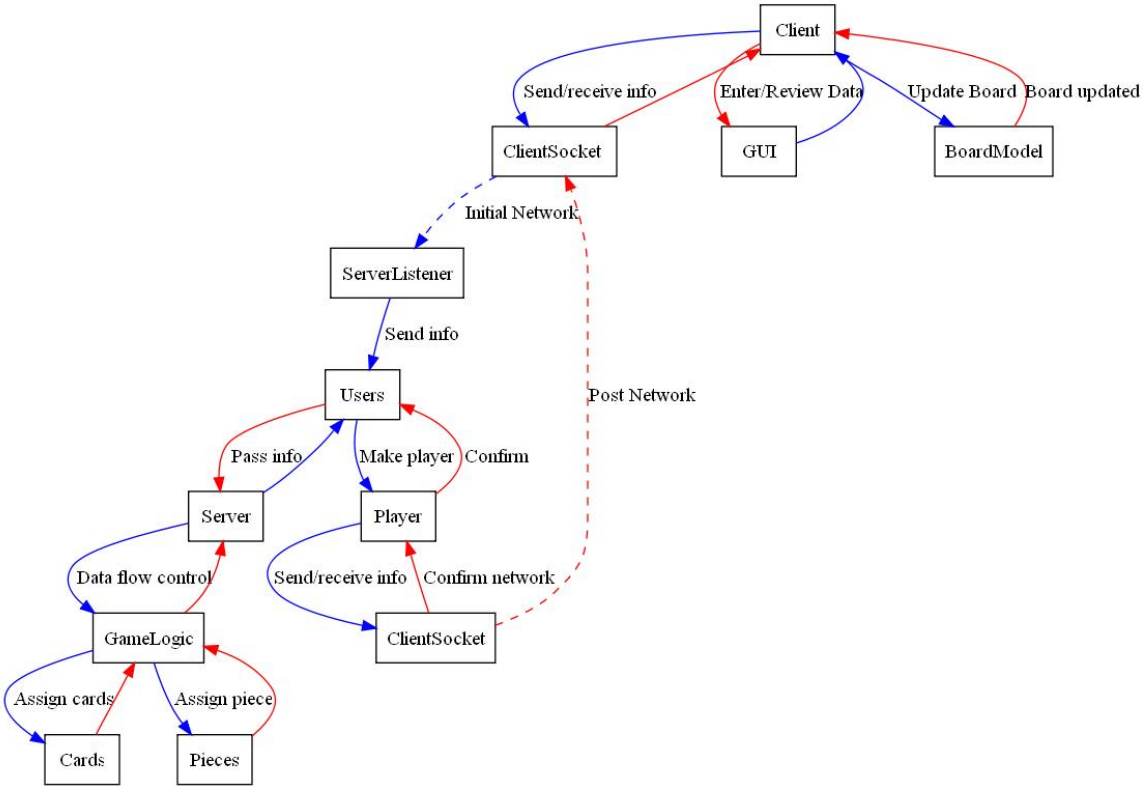
Accuse (win)
Previous Scenario: Any game-play scenario
Scenario:
The player, at any point in their turn, can elect to make an accusation and is prompted on his/her GUI to specify which character, which weapon, and which room they would like to accuse. The Server then contacts the GameLogic. It will attempt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

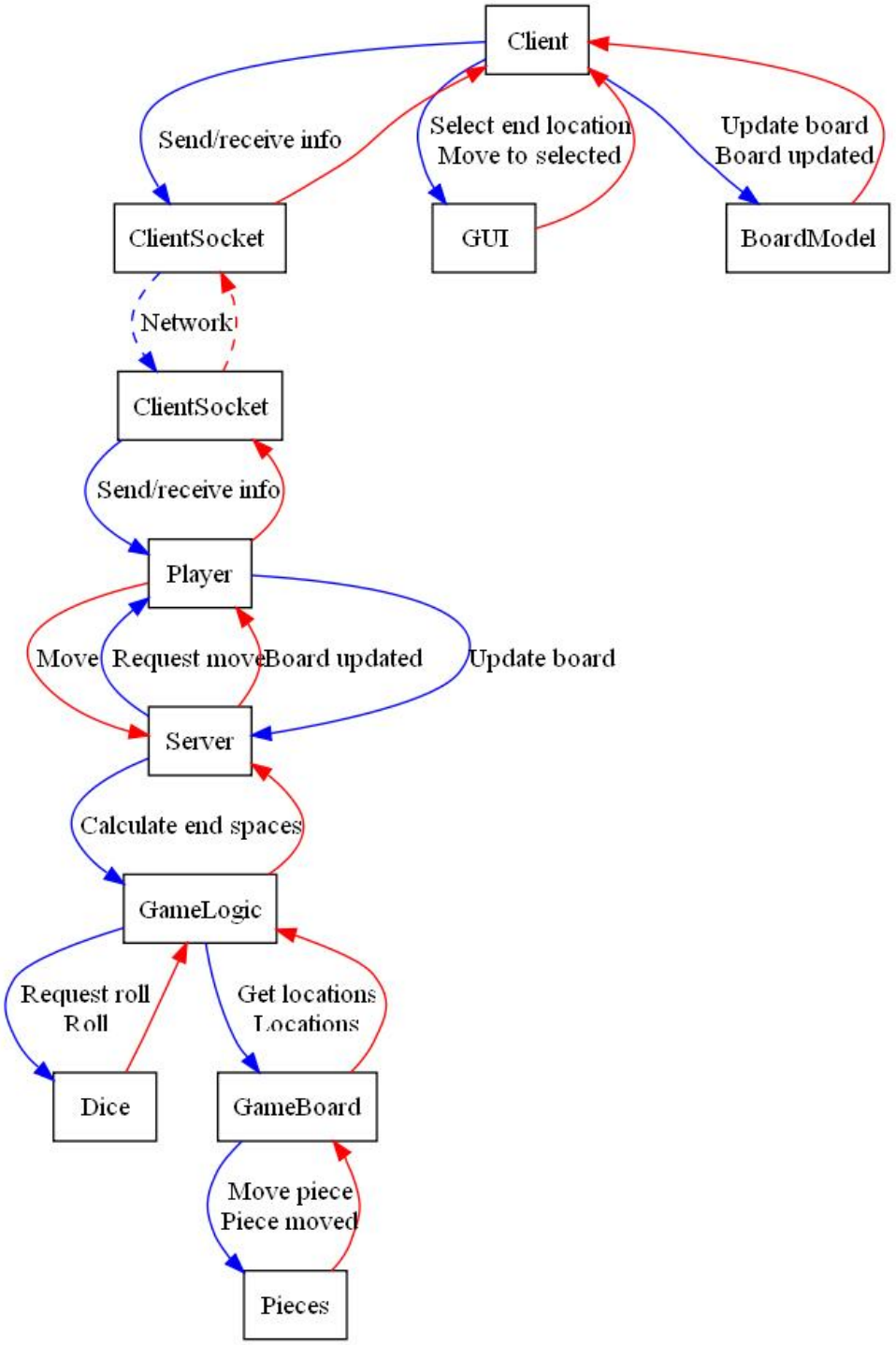
Logic which attempts to match the accusation against the secret cards. The cards do not match and the players accusation is refuted by showing him/her the appropriate secret cards. The Server then notifies all players that an accusation has been made, what the cards were and that the player has lost 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

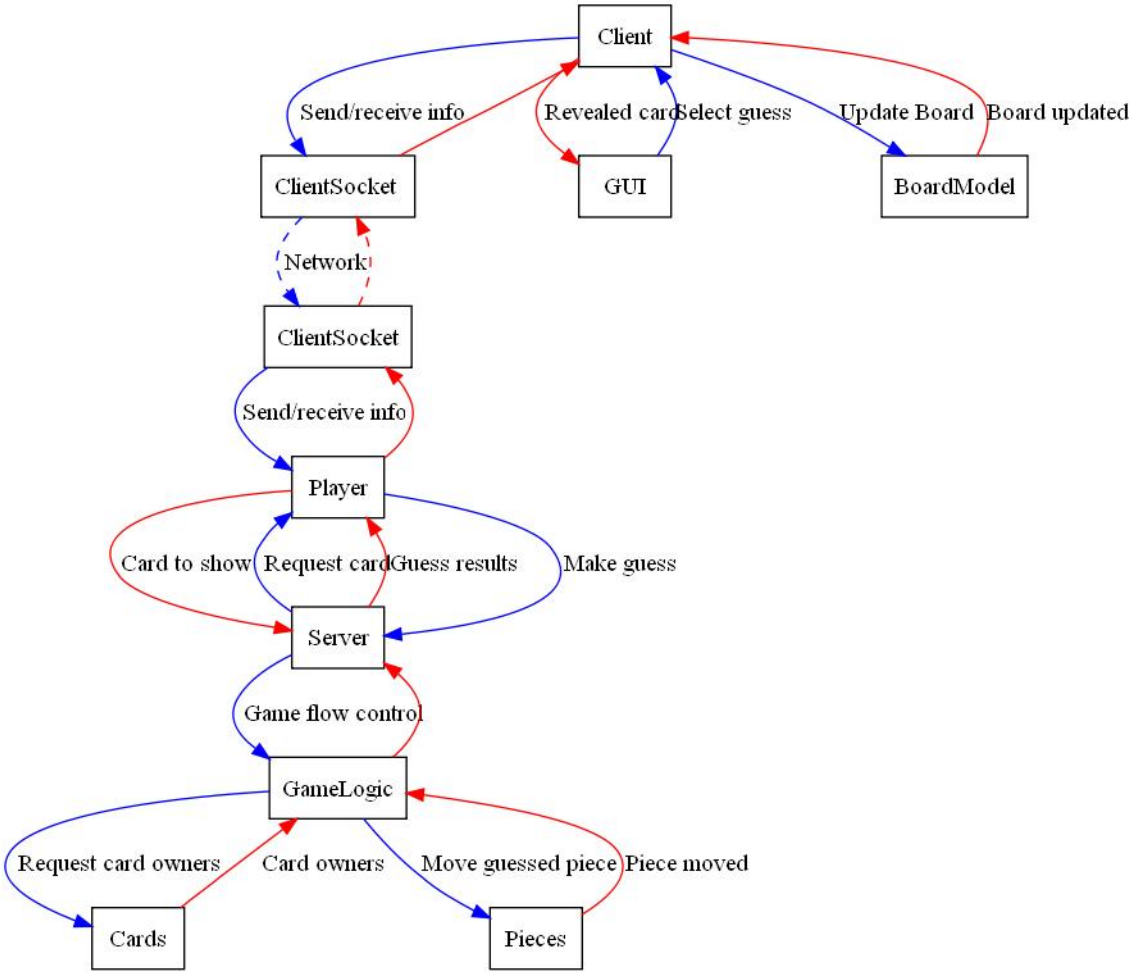
Start



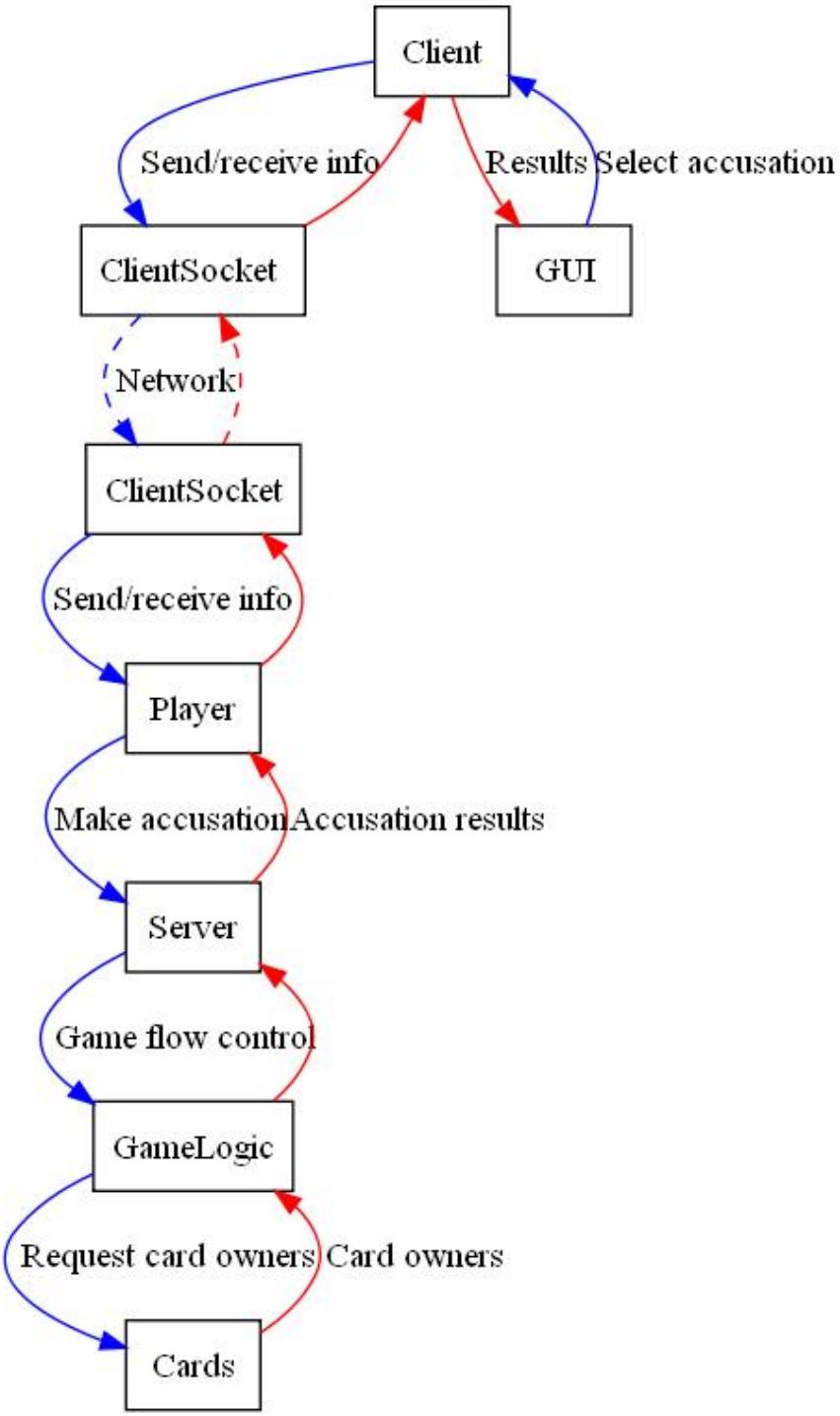
Move


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

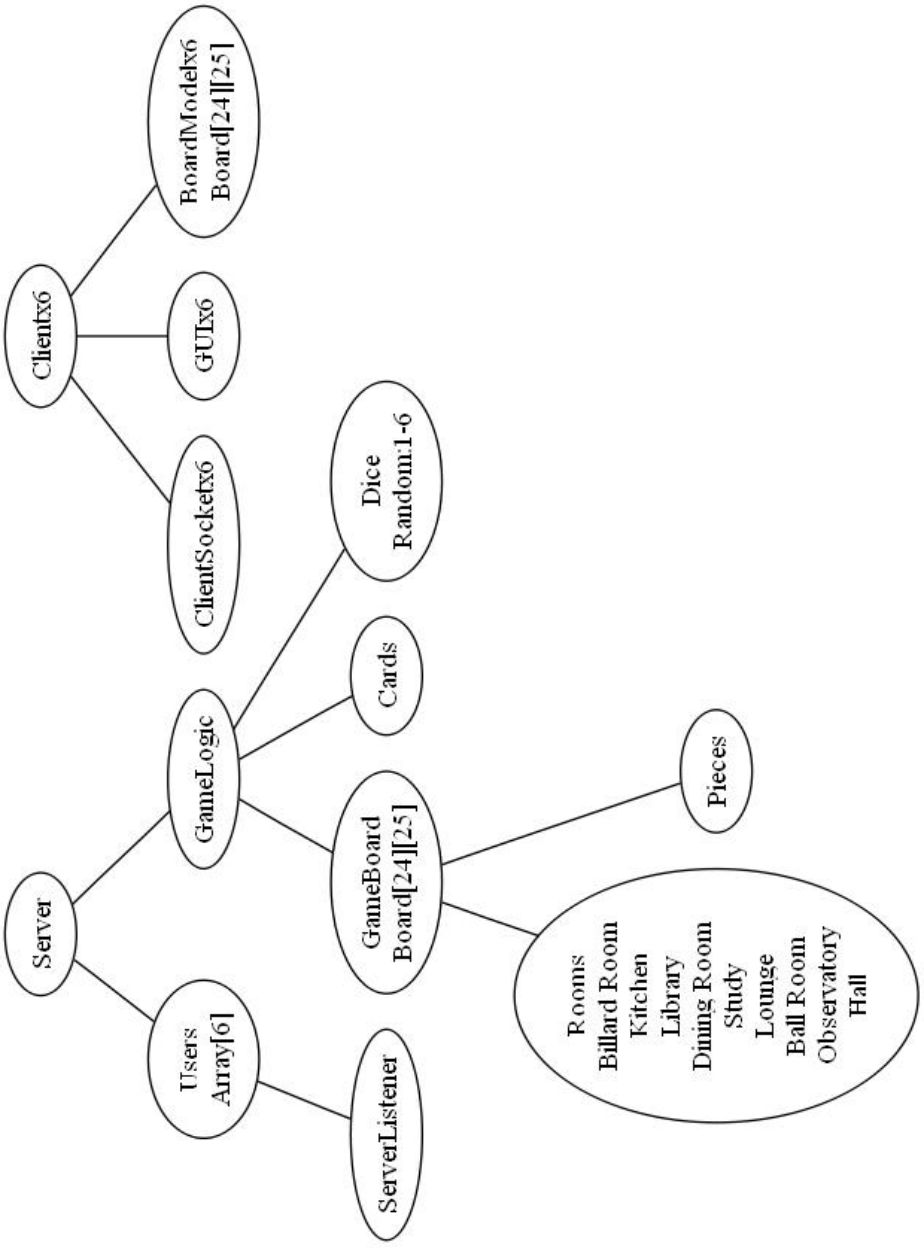

Client
Send/receive info
Revealed card
Select guess
Update Board
Board updated
ClientSocket
GUI
BoardModel
Network
ClientSocket
Send/receive info
Player
Card to show
Request card
Guess results
Make guess
Server
Game flow control
GameLogic
Request card owners
Card owners
Move guessed piece
Piece moved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

Accuse

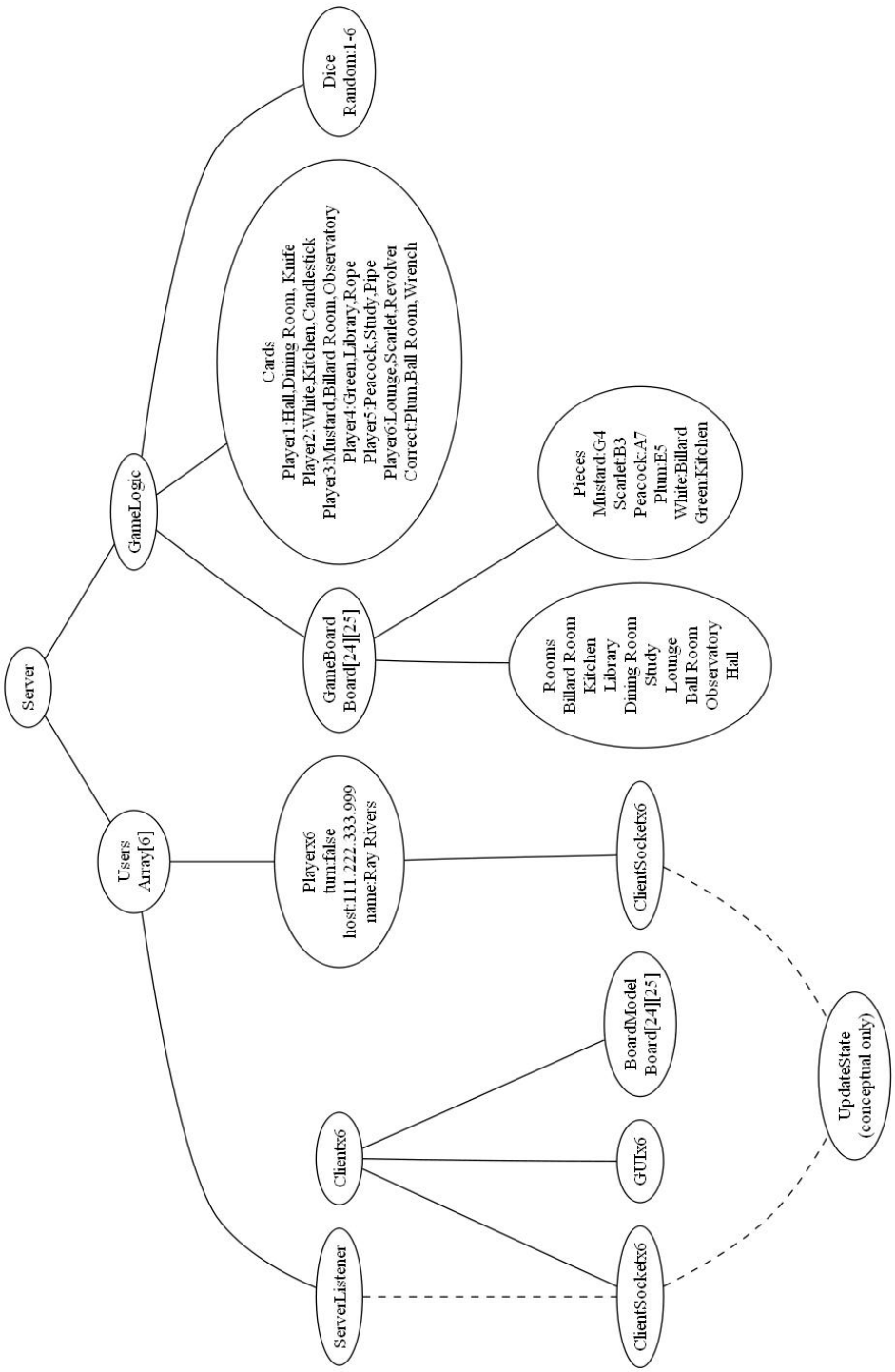

Client
Send/receive info
Results
Select accusation
ClientSocket
GUI
Network
ClientSocket
Send/receive info
Player
Make accusation
Accusation results
Server
Game flow control
GameLogic
Request card owners
Card owners
Cards

Part VII
Object Diagrams

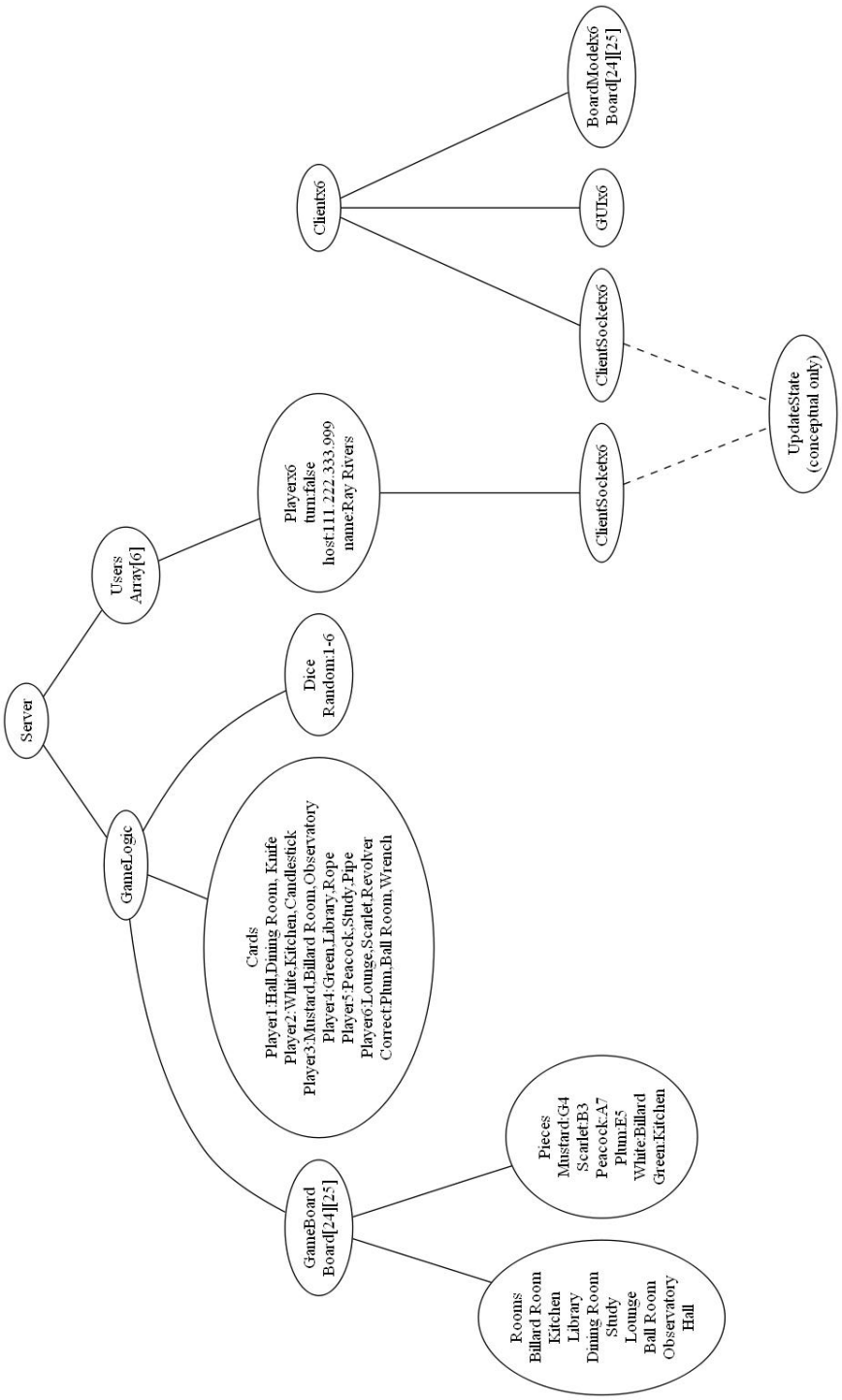
Startup



Full Model

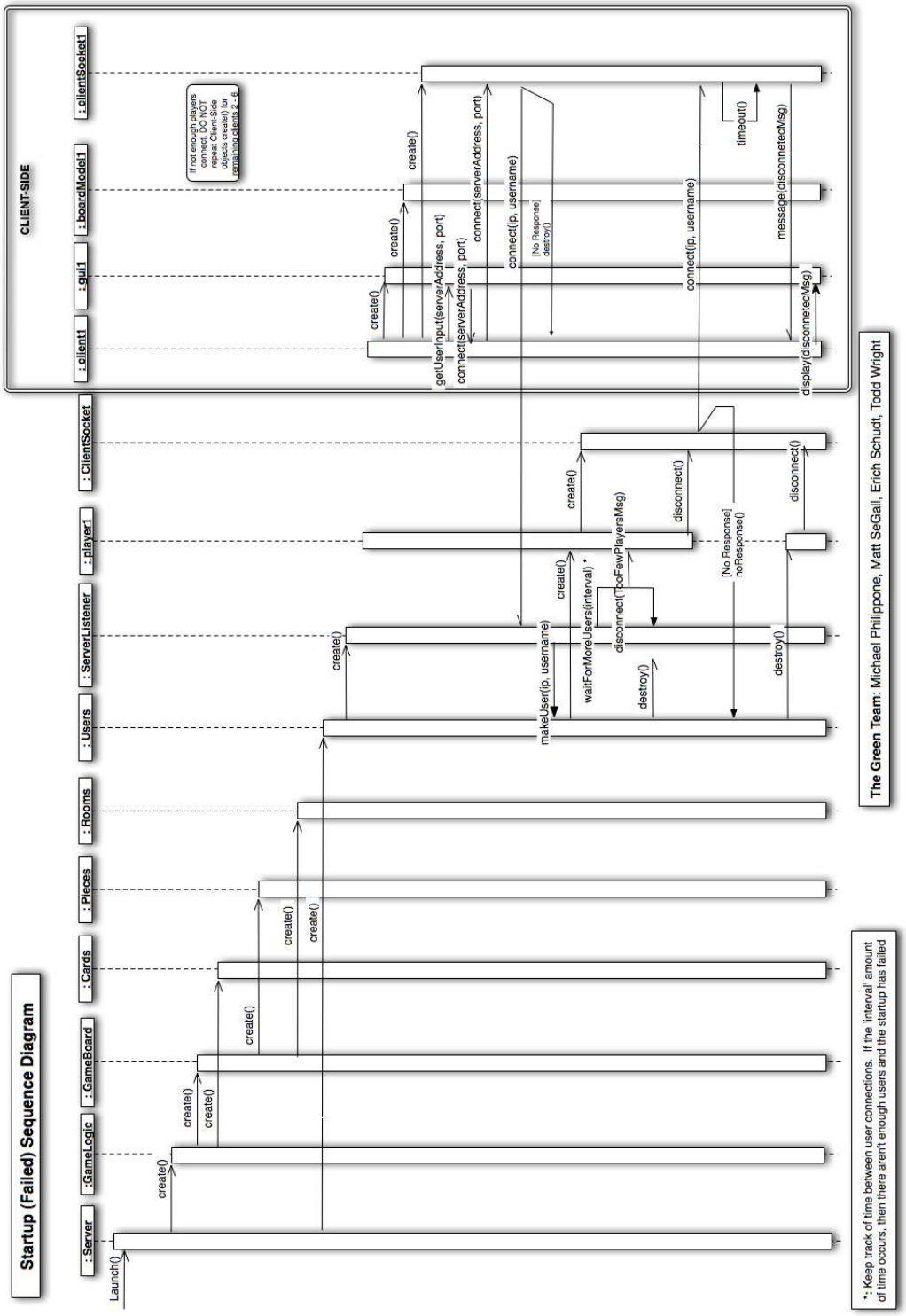

Dice
Random:1-6
Cards
Player1:Hall,Dining Room, Knife
Player2:White,Kitchen,Candlestick
Player3:Mustard,Billard Room,Observatory
Player4:Green,Library,Rope
Player5:Peacock,Study,Pipe
Player6:Lounge,Scarlet,Revolver
Correct:Plum,Ball Room,Wrench
GameLogic
Pieces
Mustard:G4
Scarlet:B3
Peacock:A7
Plum:E5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

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

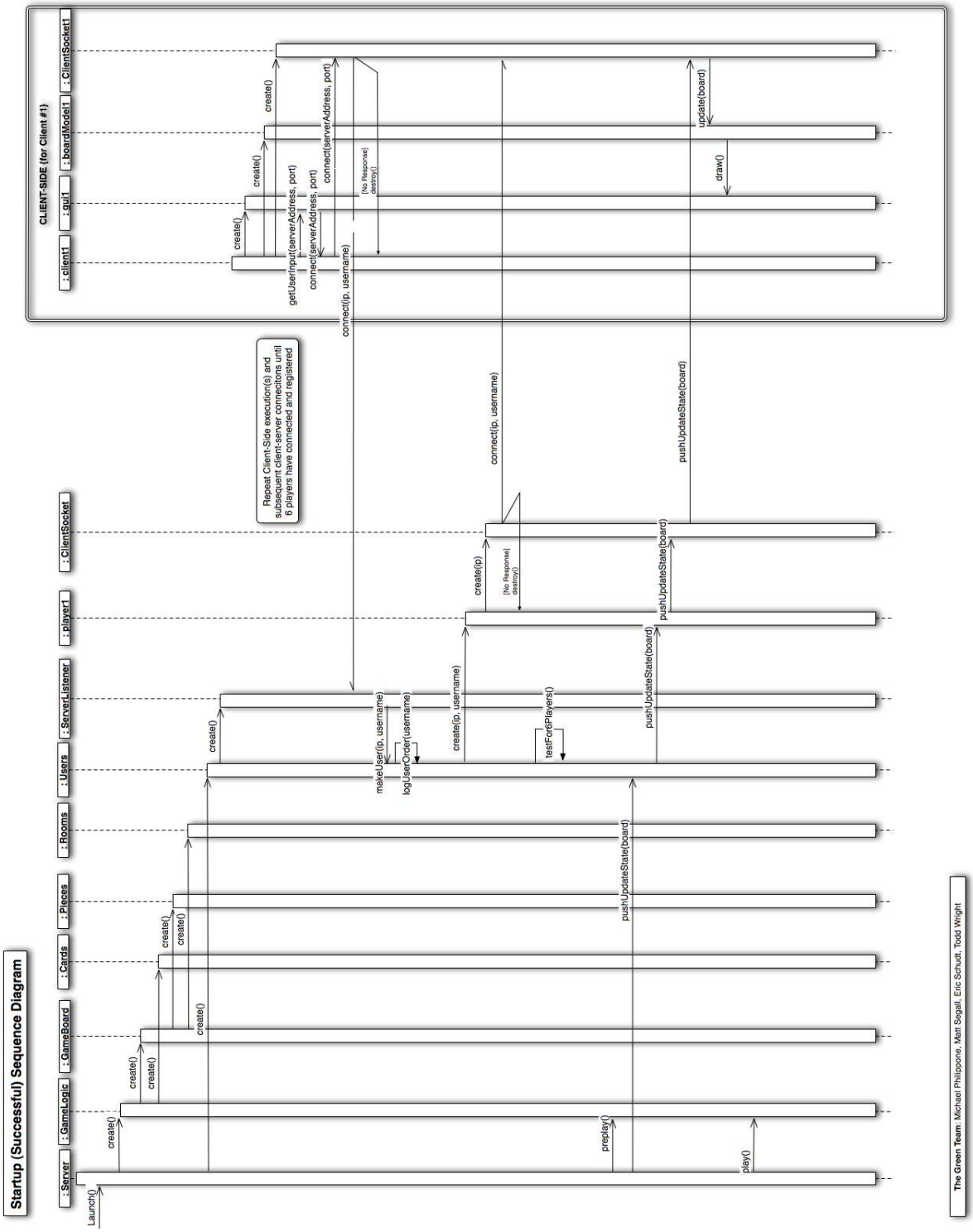


Part VIII
Sequence Diagrams

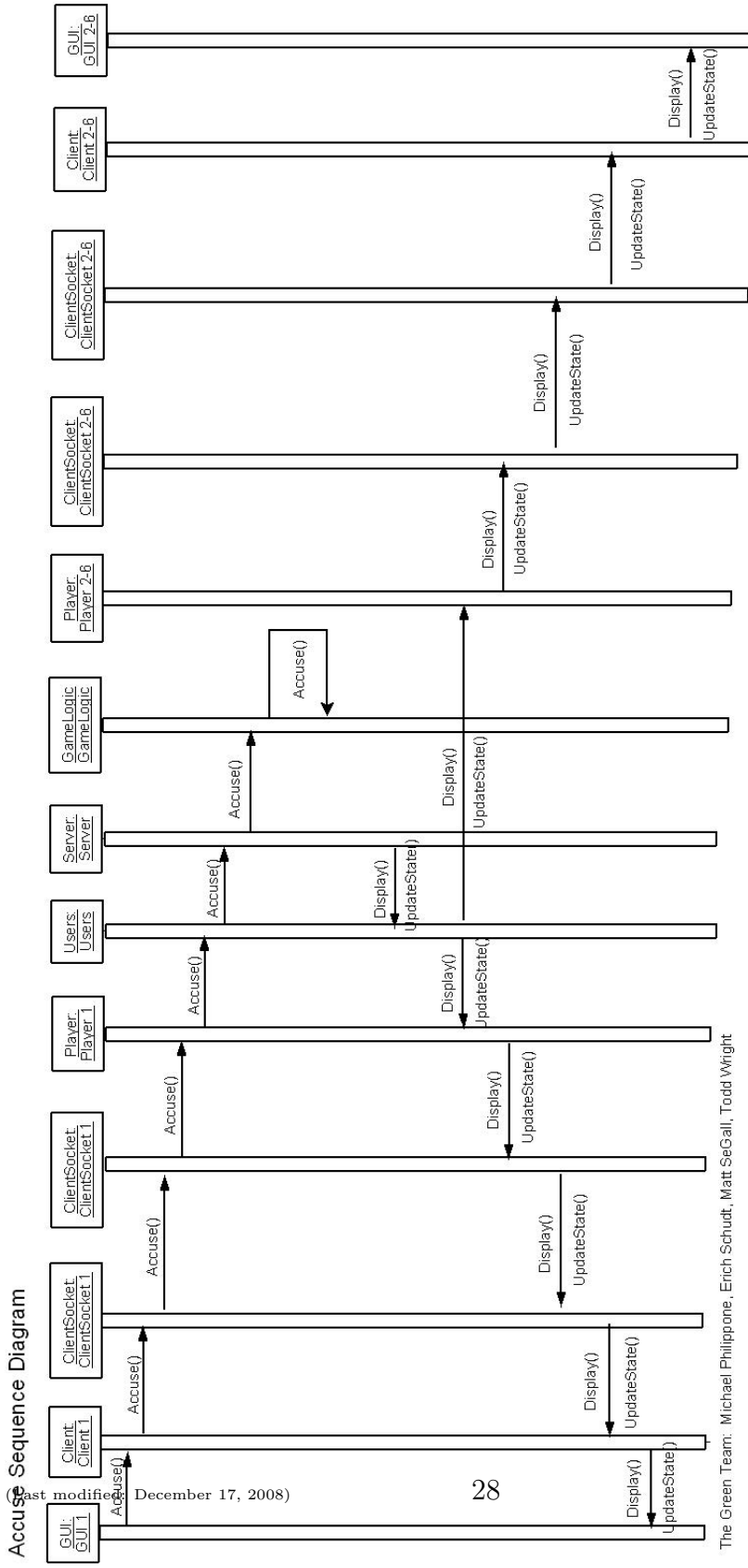
Startup (failed)



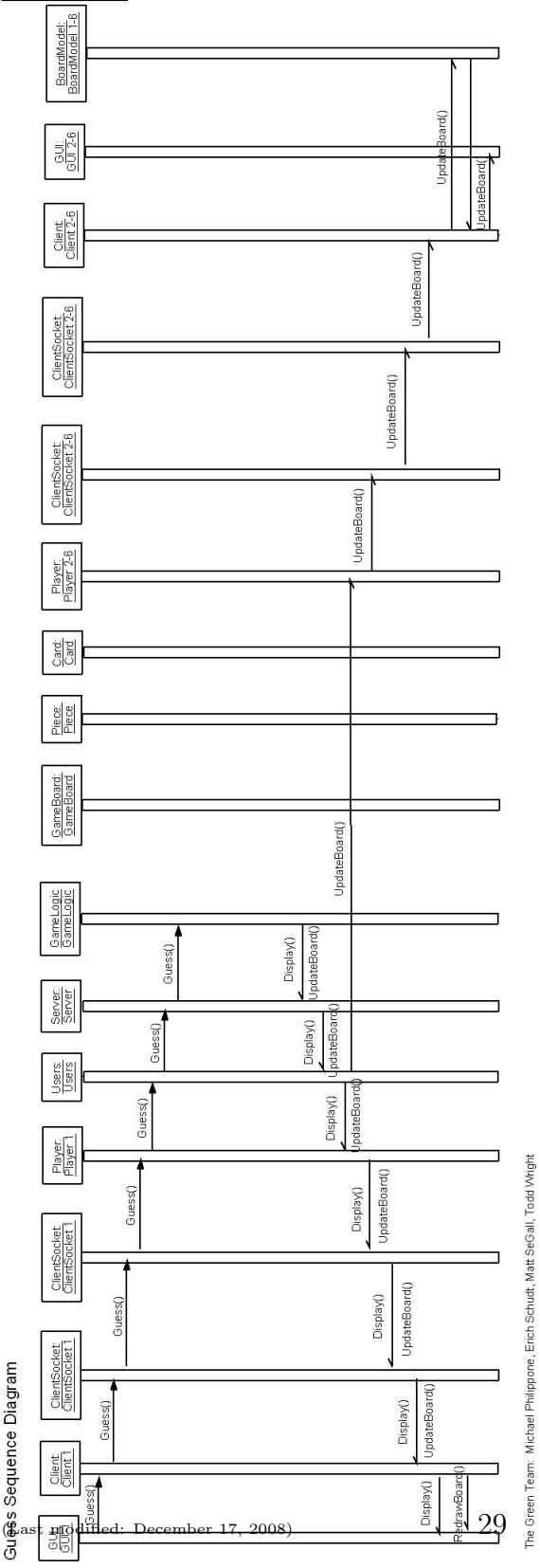
Startup (Successful)



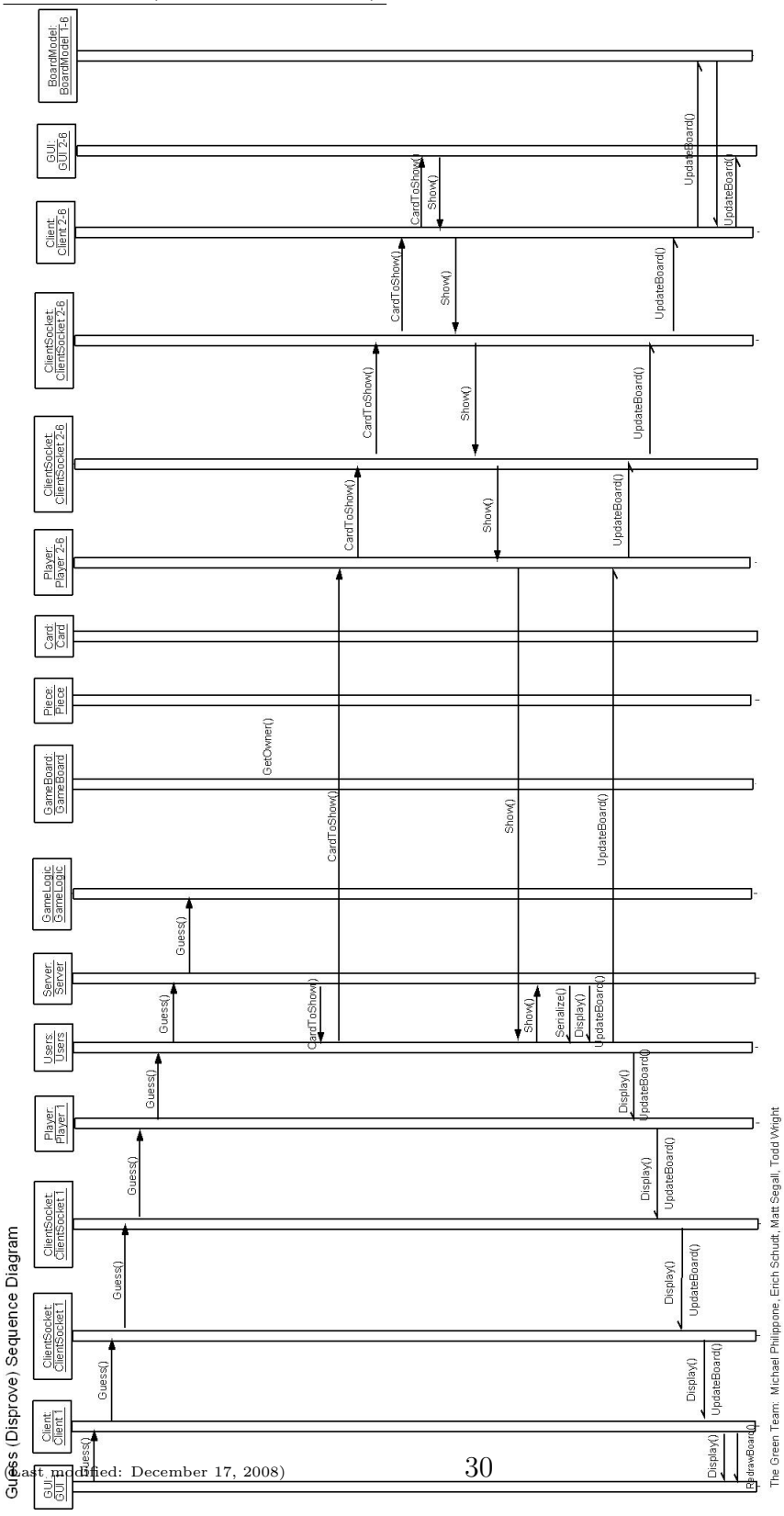
Accuse


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
ClientSocket: ClientSocket 2-6
Client: Client 2-6
GUI: GUI 2-6
Accuse()
Display()
UpdateState()
The Green Team: Michael Philippone, Erich Schudt, Matt SeGall, Todd Wright

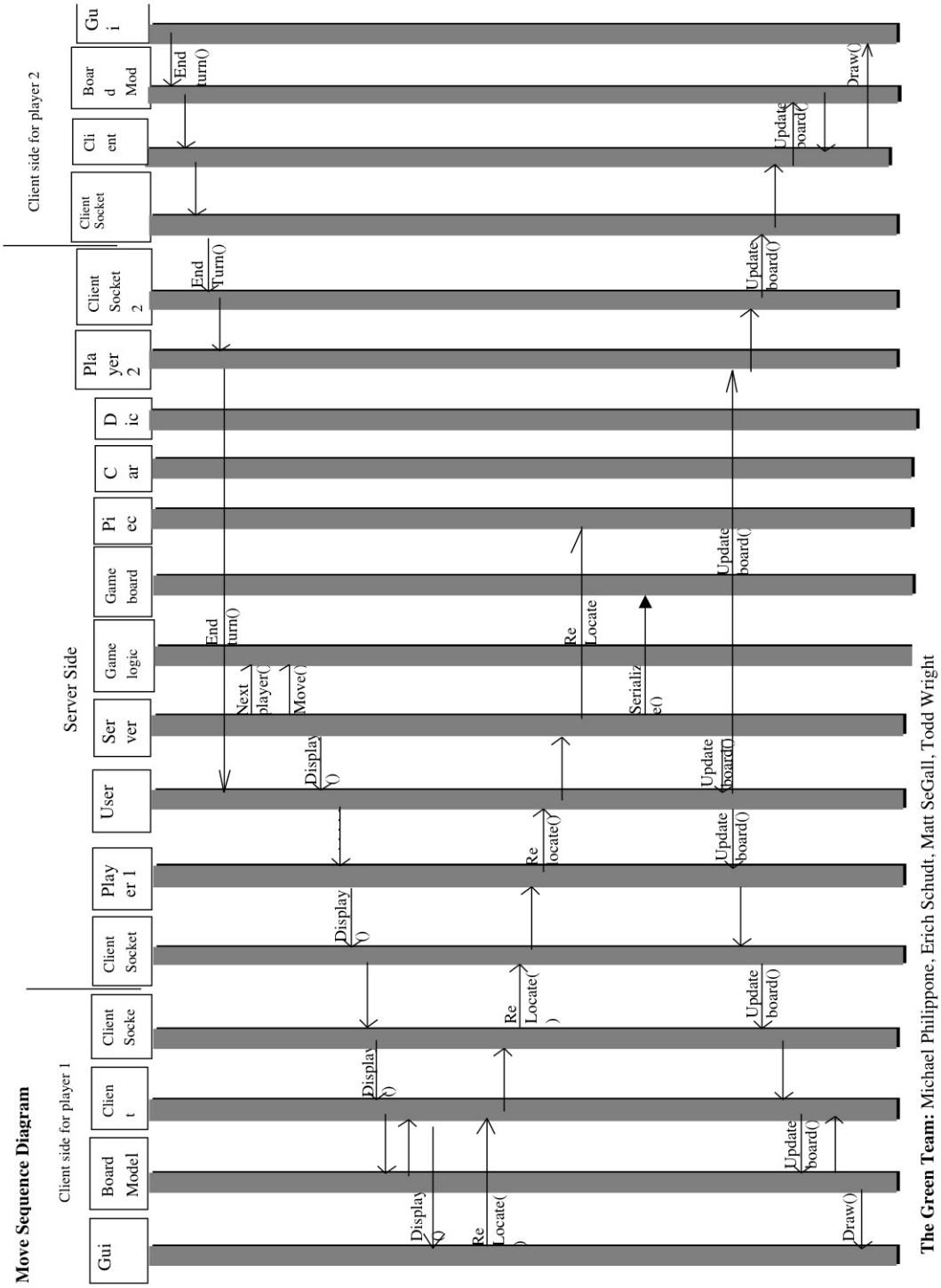
Guess


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

Guess (Disproved)



Move


Move Sequence Diagram
Client side for player 1
Client side for player 2
Server Side
Gui
Board Model
Client
Client Socke
Client Socket
Player 1
User
Server
Game logic
Game board
Piec
Car
Dic
Player 2
Client Socket 2
Client Socket
Board Model
Gui
End turn()
Next player()
Move()
Display()
Re Locate()
Update board()
Draw()
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
Test Reports

A Unit Testing

A.1 Test Cards class

```
PROGRAM STARTED AT: 12/ 17/2008 12:0:46  (mm/dd/yyy hh:mm:ss)


!THIS IS THE UNIT TEST FOR THE CARDS CLASS!
------------------------------------------
This is a test to see if the cards are being shuffled,
being assigned an owner, and if that owner is able to
be gotten.
Expected output is 3 different decks, and each owner owning
a different set of 3 cards each deck.

Shuffle #1
Card: Professor_Plum   Owner: 0
Card: Kitchen   Owner: 0
Card: Rope   Owner: 0
Card: Candlestick   Owner: 1
Card: Mr._Green   Owner: 1
Card: Dining_Room   Owner: 1
Card: Ballroom   Owner: 2
Card: Revolver   Owner: 2
Card: Pipe   Owner: 2
Card: Knife   Owner: 3
Card: Colonel_Mustard   Owner: 3
Card: Study   Owner: 3
Card: Library   Owner: 4
Card: Lounge   Owner: 4
Card: Hall   Owner: 4
Card: Observatory   Owner: 5
Card: Mrs._Peacock   Owner: 5
Card: Miss_Scarlet   Owner: 5
Card: Wrench   Owner: 6
Card: Billiard_Room   Owner: 6
Card: Miss_White   Owner: 6
```

```
--------------------
Shuffle #2
Card: Colonel_Mustard   Owner: 6
Card: Library   Owner: 3
Card: Miss_Scarlet   Owner: 6
Card: Wrench   Owner: 2
Card: Dining_Room   Owner: 2
Card: Miss_White   Owner: 1
Card: Knife   Owner: 4
Card: Mr._Green   Owner: 4
Card: Candlestick   Owner: 5
Card: Mrs._Peacock   Owner: 2
Card: Observatory   Owner: 0
Card: Kitchen   Owner: 5
Card: Ballroom   Owner: 0
Card: Revolver   Owner: 6
Card: Billiard_Room   Owner: 5
Card: Pipe   Owner: 3
Card: Hall   Owner: 3
Card: Study   Owner: 0
Card: Rope   Owner: 1
Card: Lounge   Owner: 4
Card: Professor_Plum   Owner: 1
--------------------
Shuffle #3
Card: Billiard_Room   Owner: 3
Card: Kitchen   Owner: 0
Card: Study   Owner: 2
Card: Candlestick   Owner: 1
Card: Knife   Owner: 3
Card: Ballroom   Owner: 4
Card: Rope   Owner: 1
Card: Lounge   Owner: 6
Card: Observatory   Owner: 5
Card: Colonel_Mustard   Owner: 1
Card: Professor_Plum   Owner: 3
Card: Mr._Green   Owner: 0
Card: Dining_Room   Owner: 6
Card: Miss_White   Owner: 2
Card: Miss_Scarlet   Owner: 5
Card: Wrench   Owner: 2
Card: Pipe   Owner: 6
Card: Hall   Owner: 4
```

```
Card: Revolver   Owner: 5
Card: Library   Owner: 0
Card: Mrs._Peacock   Owner: 4
```

A.2 Test Dice class

```
PROGRAM STARTED AT: 12/ 17/2008 12:5:13  (mm/dd/yyy hh:mm:ss)


!THIS IS A UNIT TEST OF THE DICE CLASS!
---------------------------------------
**This is a test of the Roll() method.**
Populate a Dice array with 1000 rolls.  Check
if each roll is not less than 1, and not greater than 6.
Expected output is that no errors occured.

Output: No errors!
```

A.3 Test GameBoard class

```
PROGRAM STARTED AT: 12/ 17/2008 12:5:42  (mm/dd/yyy hh:mm:ss)



llllllbbbWllllGbbblllllll
ltllllbgggllllgggblllltl
lllllllgglllllllllgglllllll
lllllllgglllllllllgglllllll
lllllllgglllllllllggdllllll
lllllllgdlllllllllddggggggT
lllllllgglllllllllgggggggb
ggggdgggglllllllllgglllllll
bgggggggggdggggdggdlllllll
llllllgggggggggggggllllllll
lllllllllgggggggggglllllll
lllllllllggbbbbbbgggllllll
lllllllllgdgbbbbbbgggllllll
lllllllllggbbbbbbgggggdgdgdb
lllllllllggbbbbbbgggllllll
lllllllllggbbbbbbggllllllll
bgggggdgggbbbbbbgdlllllll
Yggggggggggbbbbbbgglllllll
bgggggdggggddggggglllllll
lllllllggllllllgggggggggP
lllllllggllllllddgdgggggb
lllllllggllllllggllllllll
lllllllggllllllggllllllll
ltllllllggllllllggllllltl
lllllllRbllllllbgllllllll
```

A.4 Test GameLogic class

```
PROGRAM STARTED AT: 12/ 17/2008 12:5:54  (mm/dd/yyy hh:mm:ss)



Below are the tests for the guess() method.
Testing all combinations of true and false
GAMELOGIC: piece relocated, card owners found,
           beginning determine which to show
GAMELOGIC: some one can disprove guess, either 6, 6, or 8
GAMELOGIC: owner order is: 8 12 12
GAMELOGIC: Two belong to one player

GAMELOGIC: piece relocated, card owners found,
           beginning determine which to show
GAMELOGIC: some one can disprove guess, either 6, 6, or 8
GAMELOGIC: owner order is: 8 12 12
GAMELOGIC: Two belong to one player
null
GAMELOGIC: piece relocated, card owners found,
           beginning determine which to show
GAMELOGIC: some one can disprove guess, either 6, 6, or 8
GAMELOGIC: owner order is: 8 12 12
GAMELOGIC: Two belong to one player
null
GAMELOGIC: piece relocated, card owners found,
           beginning determine which to show
GAMELOGIC: some one can disprove guess, either 6, 6, or 8
GAMELOGIC: owner order is: 8 12 12
GAMELOGIC: Two belong to one player
8
------
GAMELOGIC: piece relocated, card owners found,
           beginning determine which to show
GAMELOGIC: some one can disprove guess, either 6, 6, or 8
GAMELOGIC: owner order is: 8 12 12
GAMELOGIC: Two belong to one player

GAMELOGIC: piece relocated, card owners found,
           beginning determine which to show
GAMELOGIC: some one can disprove guess, either 6, 6, or 8
GAMELOGIC: owner order is: 8 12 12
GAMELOGIC: Two belong to one player
```

```
null
GAMELOGIC: piece relocated, card owners found,
           beginning determine which to show
GAMELOGIC: some one can disprove guess, either 6, 6, or 8
GAMELOGIC: owner order is: 8 12 12
GAMELOGIC: Two belong to one player
null
GAMELOGIC: piece relocated, card owners found,
           beginning determine which to show
GAMELOGIC: some one can disprove guess, either 6, 6, or 8
GAMELOGIC: owner order is: 8 12 12
GAMELOGIC: Two belong to one player
8
------
-------------------
**Below are the tests for the accuse() method.
Testing all combinations of true and false**

Calling accuse() and passing in T-T-T. Expected output is true.
Output:true

Calling accuse() and passing in T-T-F. Expected output is false.
Output:false

Calling accuse() and passing in T-F-T. Expected output is false.
Output:false

Calling accuse() and passing in T-F-F. Expected output is false.
Output:false

Calling accuse() and passing in F-T-T. Expected output is false.
Output:false

Calling accuse() and passing in F-T-F. Expected output is false.
Output:false

Calling accuse() and passing in F-F-T. Expected output is false.
Output:false

Calling accuse() and passing in F-F-F. Expected output is false.
Output:false

Calling accuse() and passing in apple-banana-orange
```

```
(These cards do not exist!).  Expected output is false.
Output:false
-------------------
**Below are the tests for the nextPlayer() method.**
Player: 0 <--- This is the first player's turn.
Then we run the nextPlayer() method.
Player: 1 <--- This is the second player's turn.
Then we run the nextPlayer() method.
Player: 2 <--- This is the third player's turn.
Then we run the nextPlayer() method.
Player: 3 <--- This is the fourth player's turn.
Then we run the nextPlayer() method.
Player: 4 <--- This is the fifth player's turn.
Then we run the nextPlayer() method.
Player: 5 <--- This is the sixth player's turn.
Then we run the nextPlayer() method.
Player: 0 <--- Return back to the first player's turn.
-------------------
**Below are the tests for the move() method.**
```

A.5 Test Pieces class

```
PROGRAM STARTED AT: 12/ 17/2008 12:7:37  (mm/dd/yyy hh:mm:ss)



!THIS IS THE UNIT TEST FOR THE PIECES CLASS!
-------------------------------------------

**These are the starting positions for each player's piece.**
Miss_Scarlet's start location is:
X: 7 Y: 24

Professor_Plum's start location is:
X: 23 Y: 19

Mrs._Peacock's start location is:
X: 23 Y: 5

Mr._Green's start location is:
X: 14 Y: 0

Miss_White's start location is:
X: 9 Y: 0

Colonel_Mustard's start location is:
X: 0 Y: 17

------------------------
**This is a copy of the game board to verify that the locations
of each piece is correct.**

lllllllbbbWllllGbbblllllll
ltllllbgggllllgggblllltl
lllllllggllllllllggllllll
lllllllggllllllllggllllll
lllllllggllllllllggdllllll
lllllllgdllllllllddggggggT
lllllllggllllllllgggggggb
ggggdgggllllllllggllllll
bgggggggggdggggdggdllllll
lllllgggggggggggggllllll
llllllllgggggggggglllllll
llllllllggbbbbbgggllllll
```

```
llllllllgbbbbbbgggllllll
llllllllggbbbbbbgggggdgdb
llllllllggbbbbbbgggllllll
llllllllggbbbbbbgglllllll
bgggggdgggbbbbbbgdllllllll
Yggggggggggbbbbbbgllllllll
bgggggdgggggddggggglllllll
lllllllggllllllgggggggggP
lllllllggllllllgdgdgggggb
lllllllggllllllggllllllll
lllllllggllllllggllllllll
ltllllllggllllllggllllltl
lllllllRbllllllbgllllllll
------------------------
**This is a test of the isOccupied() method.**
**Here we pass in different coordinates
to see if that space is occupied or not.**

Pass in Miss Scarlet's position of (7, 24) into isOccupied().
Expected output is 1, because Miss Scarlet is player 1.
Output: 1

Pass in Professor Plum's position of (23, 19) into isOccupied().
Expected output is 2, because Professor Plum is player 2.
Output: 2

Pass in Mrs. Peacock's position of (23, 5) into isOccupied().
Expected output is 3, because Mrs. Peacock is player 3.
Output: 3

Pass in Mr. Green's position of (14, 0) into isOccupied().
Expected output is 4, because Mr. Green is player 4.
Output: 4

Pass in Mrs. White's position of (9, 0) into isOccupied().
Expected output is 5, because Mrs. White's is player 5.
Output: 5

Pass in Colonel Mustard's position of (0, 17) into isOccupied().
Expected output is 6, because Colonel Mustard is player 6.
Output: 6

Pass in coordinates (4, 7) which are located in the kitchen.
```

```
Expected out is 0 because no one is currently occupying that cell.
Output: 0
------------------------
**This is a test of the relocate() method.**
Professor Plum's position:
X: 23
Y: 19

*Calling relocate() and passing in (4, 7)
as Professor Plum's new coordinates.....*

Professor Plum's new/current position:
X: 4 Y: 7
------------------------
**This is a test to see if making a guess will
relocate a player to the correct room.**
Relocating Miss Scarlet to cell (0, 6) in Kitchen...
X: 0 Y: 6

Colonel Mustard's current position:
X: 0 Y: 17

Miss Scarlet implicates Colonel Mustard in a guess...
GAMELOGIC: piece relocated, card owners found,
beginning determine which to show
GAMELOGIC: some one can disprove guess, either 5, 6, or 8
GAMELOGIC: owner order is: 5 8 12
GAMELOGIC: first person ordered will disprove them

Colonel Mustard's expected position is (0,0), the first cell in Kitchen:
X: 0 Y: 0

Check if Colonel Mustard's new position is a room using
the Rooms class' isAroom() method. Expected output is true.
Output: true
```

A.6 Test Rooms class

```
PROGRAM STARTED AT: 12/ 17/2008 12:10:15  (mm/dd/yyy hh:mm:ss)


!THIS IS THE UNIT TEST FOR THE ROOMS CLASS!
-------------------------------------------
**This is a test of the isAroom() method.**
Here we run the Chambers array through the isAroom() method
to make sure that all rooms, are in fact rooms, and that the
method is actually working. Expected output is no ouput.



Now we input coordinates that were purposesly chosen
to not be part of a room.  Expected output is false.
Coordinates to be input:
X: 7 Y: 0
Output: false
--------------------
**This is a test of the isAnull() method.**
Here we run the Nulls array through the isAnull() method
to make sure that all nulls, are in fact nulls, and that the
method is actually working. Expected output is no ouput.



Now we input coordinates that were purposesly chosen
to not be a null cell.  Expected output is false.
Coordinates to be input:
X: 7 Y: 0
Output: false
```

A.7 Test message passing between Client and Server (client-side output)

```
PROGRAM STARTED AT: 12/ 17/2008 12:6:30  (mm/dd/yyy hh:mm:ss)



Message Passing and method calls Test Report

Client Startup

4 Steps to test message passing and method calls
  1) Create a ClientSocket to connect with the testing server process
  2) create a connection with another ClientSocket instance
  3) send Strings that lead to method calls in Server
  4) receive Strings to confirm methods were called
  5) close the connection

CLIENT: Step 01 Success: working so far(create connection)

Test send() in clientSocket
passed in the move protocol
Expected output:                    move 13 12  -the move method was called
CLIENT: from server  Actual output: move 13 12  -the move method was called

Test send() in clientSocket
passed in the accuse protocol
Expected output:
accuse miss_white rope study  -the move method was called
CLIENT: from server:
Actual output: accuse miss_white rope study  -the accuse method was called

Test send() in clientSocket
passed in the guess protocol
Expected output:
guess miss_scarlet revolver  -the move method was called
CLIENT: from server:
   Actual output: guess miss_scarlet revolver  -the guess method was called

Test send() in clientSocket
passed in the exit protocol
Expected output:
 exit  -the setComputerPlayer method was called
```

```
CLIENT: from server:
Actual output: exit  -the setComputerPlayer method was called

Test send() in clientSocket
passed in the endturn protocol
Expected output:                    endturn  -the nextPlayr method was called
CLIENT: from server  Actual output: endturn  -the nextPlayer method was called

CLIENT: the connection has been closed, Step 5 success

The test was a complete success. No bugs of any kind
Did not test error strings because it never occurs in the game
```

A.8 Test message passing between Client and Server (server-side output)

```
PROGRAM STARTED AT: 12/ 17/2008 12:6:25  (mm/dd/yyy hh:mm:ss)



Server Startup
4 Steps to test message passing and method calls
  1) Create a ClientSocket to connect with the testing client process
  2) receive strings that lead to method calls
  3) send confirmation to client that right strings were received
  4) close the connection
SERVER: Step 01 Success
Starting ServerSocket...
SERVER: step 2 is also good
got message 1
got message 2
got message 3
GAMELOGIC: piece relocated, card owners found,
beginning determine which to show
GAMELOGIC: some one can disprove guess, either 8, 8, or 2
GAMELOGIC: owner order is: 2 8 8
GAMELOGIC: Two belong to one player
got message 4
got message 5
SERVER: the connection has been closed, Step 4 success
Refer to client side's console for test report
```

A.9 Test message proper message receiveing in Client (client-side output)

```
PROGRAM STARTED AT: 12/ 17/2008 12:1:16  (mm/dd/yyy hh:mm:ss)



CLIENT: Step 01 Success: working so far(create connection)
got into while loop
read message 1
got into while loop
CLIENT(testerGuy): update
read message 2
got into while loop
CLIENT(testerGuy): inform: the test is working well
read message 3
got into while loop
CLIENT(testerGuy): havecard: 7;6
read message 4
got into while loop
CLIENT(testerGuy): choosecard: 7;6
read message 5
got into while loop
CLIENT(testerGuy): error reading string from server.
                   Unrecognizable socket message.
CLIENT(testerGuy): server sent: "error".
read message 6
```

A.10 Test message proper message receiveing in Client (server-side output)

```
PROGRAM STARTED AT: 12/ 17/2008 12:1:9  (mm/dd/yyy hh:mm:ss)



Starting ServerSocket...
Message Passing and method calls in Client Test Report

Server Startup

4 Steps to test message passing and method calls
  1) Create a ClientSocket to accept connecgtion from client
  2) send Strings that lead to method calls in Server
  3) receive Strings to confirm methods were called
  4) close the connection

Step 1 is a success

Test send() in clientSocket
passed in the gameover protocol
Expected output:                       -the systemShutdown method was called
SERVER: from client  Actual output:    -the systemShutdown method was called

Test send() in clientSocket
passed in the update protocol
Expected output:
                  7,0,blue,true;7,1,grey,true  - parseUpdateString method was called
SERVER: from client:
   Actual output: 7,0,blue,true;7,1,grey,true  - parseUpdateString method was called

Test send() in clientSocket
passed in the inform protocol
Expected output:
                  the test is working well  - inform method was called
SERVER: from client
   Actual output: the test is working well  - inform method was called

Test send() in clientSocket
passed in the havecard protocol
Expected output:                     7;6  -the haveCards method was called
SERVER: from client  Actual output: 7;6  -the haveCards method was called
```

```
Test send() in clientSocket
passed in the choosecard protocol
Expected output:                    7;6  -the inform method was called
SERVER: from client  Actual output: 7;6  -the inform method was called

Test send() in clientSocket
passed in an error string (no protocol)
Expected output:                    invalid string
SERVER: from client  Actual output: invalid string

SERVER: the connection has been closed, Step 5 success

The test was a complete success. No bugs of any kind
```

A.11 Network components testing (bottom-up test, client-side output)

```
PROGRAM STARTED AT: 12/ 17/2008 12:7:14  (mm/dd/yyy hh:mm:ss)



Client Startup
4 Steps to test ClientSocket (test client):
  1) Create a ClientSocket to connect with the testing server process
  2) create a connection with another ClientSocket instance
  3) communicate (send and receive packets)
  4) close the connection
CLIENT: Step 01 Success: working so far
CLIENT: sent info to server, if printed from server: Step 02 success.
CLIENT: From sever: ha ha, i got it to work
CLIENT: the connection has been closed, Step 4 success
```

A.12 Network components testing (bottom-up test, server-side output)

```
PROGRAM STARTED AT: 12/ 17/2008 12:7:4  (mm/dd/yyy hh:mm:ss)



Server Startup
5 Steps to test ClientSocket (test server):
  1) Create a ServerSocket to accept connections
  2) accept a connection from another ClientSocket instance
  3) ServerSocket spawns ClientSocket based on incoming connection
  4) Send and receive packets
  5) Close the socket connection
SERVER: Step 01 Success
SERVER: step 2 is also good
SERVER: step 3 as good as good
SERVER: time for string passing
SERVER: from client:  Ha ha, i got it to work
SERVER: If both sockets displayed their receipts, Step 4 is Successful
SERVER: the connection has been closed, Step 5 success
```

A.13 User Management components testing (client-side output)

```
PROGRAM STARTED AT: 12/ 17/2008 12:10:49  (mm/dd/yyy hh:mm:ss)



Testing the User Management Components. ( a threaded testing driver )
This is done in several steps:
  1) First, test the connection-handling abilities of the Users class
  -> spawn threads (specifically, 6) to connect to the listening Users class
  2) Once connected, mimic a Client
  -> Do so by sending one string (the username) for
   Player object creation upon connection
  3) Next, test the error handling
  -> Try to connect too many clients to the Users listener
  -> This should result in those threads labeled with AnnotherPlayer#
          never hearing back from the server process and hanging.
          (This testing driver will automatically shutdown after 2 minutes.)


FROM T_Player_0: Starting test-client Thread Player_0 ...
FROM T_Player_1: Starting test-client Thread Player_1 ...
FROM T_Player_2: Starting test-client Thread Player_2 ...
FROM T_Player_3: Starting test-client Thread Player_3 ...
FROM T_Player_4: Starting test-client Thread Player_4 ...
FROM T_Player_5: Starting test-client Thread Player_5 ...
FROM T_AnotherPlayer_0: Starting test-client Thread AnotherPlayer_0 ...
FROM T_AnotherPlayer_1: Starting test-client Thread AnotherPlayer_1 ...
FROM T_AnotherPlayer_2: Starting test-client Thread AnotherPlayer_2 ...
FROM T_AnotherPlayer_3: Starting test-client Thread AnotherPlayer_3 ...
FROM T_AnotherPlayer_4: Starting test-client Thread AnotherPlayer_4 ...
FROM T_AnotherPlayer_5: Starting test-client Thread AnotherPlayer_5 ...
FROM T_AnotherPlayer_6: Starting test-client Thread AnotherPlayer_6 ...
FROM T_Player_5:  Reading from server: This is an update for all clients!
FROM T_Player_1:  Reading from server: This is an update for all clients!
FROM T_Player_2:  Reading from server: This is an update for all clients!
FROM T_Player_3:  Reading from server: This is an update for all clients!
FROM T_Player_4:  Reading from server: This is an update for all clients!
FROM T_Player_0:  Reading from server: This is an update for all clients!
USER-MANGEMENT-COMPONENTS-TEST_CLIENT:
         Program started at 20492230, ended at 20492232 ran for 2 minutes.
         (times above in minutes)
```

A.14 User Management components testing (server-side output)

```
PROGRAM STARTED AT: 12/ 17/2008 12:10:43  (mm/dd/yyy hh:mm:ss)


Testing the User Management Components.
This is done in several steps:
  1) First, test the ability of the system to create a Users object
  2) Once the Users object is created, listen for incoming connection requests
      -> creating players until there are the required numbers
         (specifically, 6)
  3) Once the players are all connected, print out their metaData
      -> To show that they are in fact connected and to prove
          that all of the data was sent on the Socket.
  4) Then test writing a string to all of the players
  -> To simulate an update in the actual game play


Creating the Users object then listening for requests...
USERS: listening...
SERVERLISTENER: Connection detected from:
SERVERLISTENER:    ip: 127.0.0.1
SERVERLISTENER:    remote port: 52020
SERVERLISTENER:    local port: 64000
SERVERLISTENER:    username: Player_0
SERVERLISTENER: Player_0
SERVERLISTENER: Connection detected from:
SERVERLISTENER:    ip: 127.0.0.1
SERVERLISTENER:    remote port: 52021
SERVERLISTENER:    local port: 64000
SERVERLISTENER:    username: Player_1
SERVERLISTENER: Player_1
SERVERLISTENER: Connection detected from:
SERVERLISTENER:    ip: 127.0.0.1
SERVERLISTENER:    remote port: 52022
SERVERLISTENER:    local port: 64000
SERVERLISTENER:    username: Player_2
SERVERLISTENER: Player_2
SERVERLISTENER: Connection detected from:
SERVERLISTENER:    ip: 127.0.0.1
SERVERLISTENER:    remote port: 52023
SERVERLISTENER:    local port: 64000
```

```
SERVERLISTENER:    username: Player_3
SERVERLISTENER: Player_3
SERVERLISTENER: Connection detected from:
SERVERLISTENER:    ip: 127.0.0.1
SERVERLISTENER:    remote port: 52024
SERVERLISTENER:    local port: 64000
SERVERLISTENER:    username: Player_4
SERVERLISTENER: Player_4
SERVERLISTENER: Connection detected from:
SERVERLISTENER:    ip: 127.0.0.1
SERVERLISTENER:    remote port: 52025
SERVERLISTENER:    local port: 64000
SERVERLISTENER:    username: Player_5
SERVERLISTENER: Player_5
SERVERLISTENER: Required number of players connected.
USERS: made it to the end of the user's constructor
Players connected...
Players' metadata:
T_T_Player_0 Player's name: T_Player_0
T_T_Player_0 Player's piece: Miss_Scarlet
T_T_Player_0 Player's connection order: -1
T_T_Player_1 Player's name: T_Player_1
T_T_Player_1 Player's piece: Professor_Plum
T_T_Player_1 Player's connection order: -1
T_T_Player_2 Player's name: T_Player_2
T_T_Player_2 Player's piece: Mrs._Peacock
T_T_Player_2 Player's connection order: -1
T_T_Player_3 Player's name: T_Player_3
T_T_Player_3 Player's piece: Mr._Green
T_T_Player_3 Player's connection order: -1
T_T_Player_4 Player's name: T_Player_4
T_T_Player_4 Player's piece: Miss_White
T_T_Player_4 Player's connection order: -1
T_T_Player_5 Player's name: T_Player_5
T_T_Player_5 Player's piece: Colonel_Mustard
T_T_Player_5 Player's connection order: -1
Writing to all players...


Done. Exiting.
```

B Integration Testing

B.1 Testing Clients' connection to Server process:

```
PROGRAM STARTED AT: 12/ 17/2008 13:8:47  (mm/dd/yyy hh:mm:ss)



This test is to show that the required number of clients
   can connect to a running server instance and properly set themselves
   for a game to begin.
This includes:
  - Connecting and registering with the Server process
      (Sending username)
      (receiveing information about the player's assigned game piece)
  - Obtaining a current model of the board to display on the GUI.
  - Obtaining and displaying the hand of 3 cards the player has been dealt
  - The first player should have a roll occurr
  - All other player should be alerted to the roll's value
  - The first player's board should light up with the appropriate ending spaces

That is the extent of the preplay.

The console out put from this test run, in conjunciton with the GUI
  Panel updates will prove that the above conditions occurred
  (or did not occurr)

Please also refer to the Server process' "Clue_LOG.txt"
    file for more information on connection and communication conditions

The Rest of this file is the output from the requisite number
    of Client threads and and their subsequent Game
    updates and client-server communications

NOTE: this test turns off after 2 minutes.
      Since no one is playing the game, there will be no progress.

*************************************************************************


BEGIN
*************************************************************************
```

```
CLIENT(mphilip1_0): username: mphilip1_0
CLIENT(mphilip1_1): username: mphilip1_1
CLIENT(mphilip1_2): username: mphilip1_2
CLIENT(mphilip1_3): username: mphilip1_3
CLIENT(mphilip1_4): username: mphilip1_4
CLIENT(mphilip1_5): username: mphilip1_5
CLIENT(mphilip1_5): inform: You have been assigned the piece"Colonel_Mustard"
CLIENT(mphilip1_3): inform: You have been assigned the piece"Mr._Green"
CLIENT(mphilip1_0): inform: You have been assigned the piece"Miss_Scarlet"
CLIENT(mphilip1_1): inform: You have been assigned the piece"Professor_Plum"
CLIENT(mphilip1_2): inform: You have been assigned the piece"Mrs._Peacock"
CLIENT(mphilip1_4): inform: You have been assigned the piece"Miss_White"
CLIENT(mphilip1_5): havecard: Kitchen;Professor_Plum;Ballroom
CLIENT(mphilip1_3): havecard: Mrs._Peacock;Candlestick;Knife
CLIENT(mphilip1_2): havecard: Dining_Room;Colonel_Mustard;Library
CLIENT(mphilip1_4): havecard: Lounge;Rope;Miss_Scarlet
CLIENT(mphilip1_0): havecard: Hall;Wrench;Mr._Green
CLIENT(mphilip1_1): havecard: Revolver;Observatory;Study
CLIENT(mphilip1_2): update
CLIENT(mphilip1_0): update
CLIENT(mphilip1_1): update
CLIENT(mphilip1_3): update
CLIENT(mphilip1_4): update
CLIENT(mphilip1_5): update
CLIENT(mphilip1_3): inform: mphilip1_0 has rolled a 5
CLIENT(mphilip1_0): inform: mphilip1_0 has rolled a 5
CLIENT(mphilip1_2): inform: mphilip1_0 has rolled a 5
CLIENT(mphilip1_1): inform: mphilip1_0 has rolled a 5
CLIENT(mphilip1_5): inform: mphilip1_0 has rolled a 5
CLIENT(mphilip1_4): inform: mphilip1_0 has rolled a 5
CLIENT(mphilip1_0): update
CLIENTTEST: started at 1229537327711, ended at 1229537447717 ran for 2 minutes.
```

B.2 Sample Server output log

```
SERVER: Debugging mode set to 'ON'.
SERVER: All messages will be prefaced with the originating component in caps.
SERVER: program started at 12/ 17/2008 13:7:48  (mm/dd/yyy hh:mm:ss).

USERS: listening...
SERVERLISTENER: Connection detected from:
SERVERLISTENER:    ip: 127.0.0.1
SERVERLISTENER:    remote port: 52121
SERVERLISTENER:    local port: 64000
SERVERLISTENER:    username: mphilip1_0
PLAYER: Player created.
    IP: 127.0.0.1
    Username: mphilip1_0
    Assigned piece: Miss_Scarlet.
SERVERLISTENER: mphilip1_0
SERVERLISTENER: There are now 1 users connected.
SERVERLISTENER: Connection detected from:
SERVERLISTENER:    ip: 127.0.0.1
SERVERLISTENER:    remote port: 52122
SERVERLISTENER:    local port: 64000
SERVERLISTENER:    username: mphilip1_1
PLAYER: Player created.
    IP: 127.0.0.1
    Username: mphilip1_1
    Assigned piece: Professor_Plum.
SERVERLISTENER: mphilip1_1
SERVERLISTENER: There are now 2 users connected.
SERVERLISTENER: Connection detected from:
SERVERLISTENER:    ip: 127.0.0.1
SERVERLISTENER:    remote port: 52123
SERVERLISTENER:    local port: 64000
SERVERLISTENER:    username: mphilip1_2
PLAYER: Player created.
    IP: 127.0.0.1
    Username: mphilip1_2
    Assigned piece: Mrs._Peacock.
SERVERLISTENER: mphilip1_2
SERVERLISTENER: There are now 3 users connected.
SERVERLISTENER: Connection detected from:
SERVERLISTENER:    ip: 127.0.0.1
SERVERLISTENER:    remote port: 52124
```

```
SERVERLISTENER:    local port: 64000
SERVERLISTENER:    username: mphilip1_3
PLAYER: Player created.
    IP: 127.0.0.1
    Username: mphilip1_3
    Assigned piece: Mr._Green.
SERVERLISTENER: mphilip1_3
SERVERLISTENER: There are now 4 users connected.
SERVERLISTENER: Connection detected from:
SERVERLISTENER:    ip: 127.0.0.1
SERVERLISTENER:    remote port: 52125
SERVERLISTENER:    local port: 64000
SERVERLISTENER:    username: mphilip1_4
PLAYER: Player created.
    IP: 127.0.0.1
    Username: mphilip1_4
    Assigned piece: Miss_White.
SERVERLISTENER: mphilip1_4
SERVERLISTENER: There are now 5 users connected.
SERVERLISTENER: Connection detected from:
SERVERLISTENER:    ip: 127.0.0.1
SERVERLISTENER:    remote port: 52126
SERVERLISTENER:    local port: 64000
SERVERLISTENER:    username: mphilip1_5
PLAYER: Player created.
    IP: 127.0.0.1
    Username: mphilip1_5
    Assigned piece: Colonel_Mustard.
SERVERLISTENER: mphilip1_5
SERVERLISTENER: There are now 6 users connected.
SERVERLISTENER: Required number of players connected.
USERS: made it to the end of the user's constructor
SERVER: made it to the end of server's createservercomponents method
SERVER: made it to the preplay method!
SERVER: true (result of inform piece name command sent to user: mphilip1_0)
SERVER: true (result of inform piece name command sent to user: mphilip1_1)
SERVER: true (result of inform piece name command sent to user: mphilip1_2)
SERVER: true (result of inform piece name command sent to user: mphilip1_3)
SERVER: true (result of inform piece name command sent to user: mphilip1_4)
SERVER: true (result of inform piece name command sent to user: mphilip1_5)
SERVER: true (result of havecard command sent to user: mphilip1_0)
SERVER: true (result of havecard command sent to user: mphilip1_1)
SERVER: true (result of havecard command sent to user: mphilip1_2)
```

```
SERVER: true (result of havecard command sent to user: mphilip1_3)
SERVER: true (result of havecard command sent to user: mphilip1_4)
SERVER: true (result of havecard command sent to user: mphilip1_5)
SERVER: true (result of update command sent to user: mphilip1_0)
SERVER: true (result of update command sent to user: mphilip1_1)
SERVER: true (result of update command sent to user: mphilip1_2)
SERVER: true (result of update command sent to user: mphilip1_3)
SERVER: true (result of update command sent to user: mphilip1_4)
SERVER: true (result of update command sent to user: mphilip1_5)
SERVER: TOP SECRET cards are:
        Miss_White did it in the Billiard_Room with the Pipe
GAMELOGIC: dice roll value = 5
SERVER: received "testingshutdown" from player: mphilip1_0 (while loop bottom)
SERVER: System shutting down with exit code "4".
```

Part XIV
Meeting Minutes

Meeting 01

Green Team Minutes
TIME / PLACE: 07 SEPT 2008, 1 PM
IN ATTENDANCE: Todd, Eric, Matt (Michael was home Friday to Monday)
FACILITATOR: Matt
SECRETARY: Matt
BODY:

1) Brainstorm possible scenarios.
2) Followed with write up of scenarios.
3) Brainstormed ideas and other possible scenarios.
4) Discussed our coding ability.

ASSIGNMENTS:

1) Matt, type up scenarios
2) Todd, edit scenarios
3) Eric, turn in finalized scenarios
4) Mike, catch up

END TIME: 1:15 pm
NEXT MEETING: Following weekend.

Meeting 02

Green Team Minutes
TIME / PLACE: 14 SEPT 2008, 4 PM, McDonald Lounge: Phillips Hall (Campus Center)
IN ATTENDANCE: Mike, Eric, Todd, Matt
FACILITATOR: Matt
SECRETARY: Mike
BODY:

1) Discussed our upcoming due dates and responsibilities.
2) Discussed the rules for playing the full version of Clue
3) Discussed and worked on re-doing the work for "Informal Scenarios"
   --> worked as a group to fix the problems & tweak them to fit real Clue rules
4) Discussed and worked up the "Supported Activities List" for our game
5) Discussed and signed up for (online form) the "John Barr Hour" weekly meetings
   --> requested 3:30 PM every FRIDAY

ASSIGNMENTS:

1) All members of group should re-read today's work and if necessary submit changes by Tuesday {16 SEPT } night. (before Wednesday {17 SEPT } due-date)
2) Mark Calendars with weekly "John Barr Hour Time"
   -> 3:30 PM every FRIDAY

Meeting 03

Green Team Minutes
TIME / PLACE: 25 SEPT 2008, 12 NOON, Williams 303
IN ATTENDANCE: Mike, Eric, Todd, Matt
FACILITATOR: Matt
SECRETARY: Mike
BODY:

1) Review the basics of UCCD
   --> Review class slideshow, lecture notes, etc.
2) outline all UCs for our Clue Game
   --> whiteboard, who is doing what, what Informal Scenarios apply to which UCs, etc.

ASSIGNMENTS:

1) EVERYONE should read up on sockets and GUI programming
2) Everyone work on their assigned UCs (and Diagrams!) and email them to Mike, who will zip them and submit them via blackboard by sun night.
3) Also, make sure to SEND ALL when emailing, so we can all vet one another's work
4) NEXT MEETING: TBD

Meeting 04

Green Team Minutes
TIME / PLACE: 21 SEPT 2008, 4 PM, McDonald Lounge: Phillips Hall (Campus Center)

IN ATTENDANCE: Mike, Matt, Eric, Todd

FACILITATOR: Todd

SECRETARY: Mike

BODY:

1) Went over class listing and wrote up what we need.
   -- listed the classes and drew them with Omnigraffle (<- Michael)
2) talked about implementation of some of the server-side game state representations
3) talked a bit about Socket programming

ASSIGNMENTS:

1) Everyone should make time to stop by Williams 309 and read up on Socket programming.
2) Everyone should do the JAVA Swing tutorial that John linked to in his PPT slide
3) Everyone should check out the group work repository at
   http://students.ithaca.edu/~mphilip1/SoftEng/
4) NEXT MEETING: Thursday 25 SEPT 2008, 12 Noon Williams 308

Meeting 05

Green Team Minutes
TIME / PLACE: 25 SEPT 2008, 12 NOON, Williams 303
IN ATTENDANCE: Mike, Eric, Todd, Matt
FACILITATOR: Matt
SECRETARY: Mike
BODY:

1) Review the basics of UCCD
   --> Review class slideshow, lecture notes, etc.
2) outline all UCs for our Clue Game
   --> whiteboard, who is doing what, what Informal Scenarios apply to which UCs, etc.

ASSIGNMENTS:

1) EVERYONE should read up on sockets and GUI programming
2) Everyone work on their assigned UCs (and Diagrams!) and email them to Mike, who will zip them and submit them via blackboard by sun night.
3) Also, make sure to SEND ALL when emailing, so we can all vet one another's work
4) NEXT MEETING: TBD

Meeting 06

Green Team Minutes

TIME / PLACE: 12 NOON, Williams 309

IN ATTENDANCE: Todd, Matt, Mike

FACILITATOR: Matt

SECRETARY: Mike

BODY:

1) Go over formal scenarios requirements
    a) draw them up (see use cases)
    b) parse them out to each group member

ASSIGNMENTS:

1) Mike: Move--room, Move--tunnel, Move--player's piece @ end, Move--standard
2) Todd: Exception--network, CreateGame--valid, CreateGame--wait for more players
3) Matt: Update, Start--valid, Start--invalid(too few players), Roll
4) Eric: Guess--disproved, Guess--notDisproved

NEXT MEETING: Meeting on Sunday, Usual place and time (4PM Klingenstein Lounge)

Meeting 07

Green Team Minutes
TIME / PLACE: 1 PM Klingenstein Lounge (Campus Center)
IN ATTENDANCE: Mike, Matt, Todd (Eric had surprise commitment, unable to attend mtg)
FACILITATOR: Mike
SECRETARY: Mike
BODY:

1) Discuss Formal scenarios
    a) hash out who did what, how many there are, discuss issues with implementation
2) fix Class Listing assignments
    a) add / remove classes in accordance with John's proof-readings
    b) re-work client <--> server communication
    c) discuss implementation of guesses, player state, client-state updates
3) work out Guess and Accuse formal scenarios

ASSIGNMENTS:

1) If you have not done so, please(!) email mike your Formal Scenarios in Microsoft Word format so he can compile them all and

****Don't forget, The John Barr Hour has moved this week to Thursday at 11AM****

NEXT MEETING: *TBD*

Meeting 08

Green Team Minutes
TIME / PLACE: Klingenstein Lounge
IN ATTENDANCE: Matt, Mike, Eric, Todd
FACILITATOR: Matt
SECRETARY: Mike
BODY:

1) Discussed previous grade on returned "Class Listing assignment (Revision 01)"
2) Checked in on class collaboration diagram progress
3) worked out the Object diagrams
4) discussed implementation of threading in the project
   -> Re-structured the Server-side "PlayerClass" to be threaded and have a socket for client communication

ASSIGNMENTS:

1) Mike do remaining class collaboration diagram and zip this week's deliverables by noon tomorrow (13 OCT)

NEXT MEETING: *TBD*

Meeting 09

Green Team Minutes
TIME / PLACE: Williams 308, 23 OCT, 2:00 PM
IN ATTENDANCE: ALL
FACILITATOR: Matt
SECRETARY: Mike
BODY:

1) Discuss midterm (yikes!)

2) Discuss upcoming deliverable
   -- Turn our 7 formal scenarios into sequence diagrams
      -- Startup (success) & Startup (fail) --> **Mike**
      -- Move --> **Eric**
      -- Guess (disproved) & Guess (proved) --> **Matt**
      -- Accuse (win) & Accuse (loss) --> **Todd**

3) Update all past work to compile into "design book"
   -- Informal Scenarios --> **Eric**
   -- Class List --> **Mike**
   -- Supported Activities --> **Todd**

4)

ASSIGNMENTS:

1) See right-hand side of arrows above for who is in charge of what for sunday's mtg.

NEXT MEETING: **1 PM**, Klingenstein Lounge, **Sun. 26 OCT**

Meeting 10

Green Team Minutes
TIME / PLACE: 1PM, 26 OCT, Klingenstein Lounge
IN ATTENDANCE: ALL
FACILITATOR: Eric
SECRETARY: Mike
BODY:

1) Discuss Sequence Diagrams
    a. Work out a few examples (startup, move)
2) Discussed structure of GameLogic class

ASSIGNMENTS:

1) **Everyone**:
    a. Finish your sequence diagrams
    b. Email to Mike so he can zip and submit them
    c. Finish our design phase updates and email them to mike
2) **Mike**:
    d. Update the design phase documents, print them all into design book
    e. Publish to repository

NEXT MEETING: *TBD*

Meeting 11

Green Team Minutes
TIME / PLACE: 3PM 2 NOV 2008, Klingenstein Lounge
IN ATTENDANCE: ALL
FACILITATOR: Erich
SECRETARY: Mike
BODY:

1) Implementation Plan
   a. Read through PPT on Implementation Plans
   b. Discuss Phases
   c. Discuss project implmentation timeline

ASSIGNMENTS:

1) Find a good time to meet again this week
2) Fix up those issues that you can on anything that you have worked thus far

NEXT MEETING: ***TBD***

Meeting 12

Green Team Minutes
TIME / PLACE: Williams 308, 11AM, 4 NOV 2008
IN ATTENDANCE: Mike, Erich, Matt, Todd
FACILITATOR: Matt
SECRETARY: Mike
BODY:

1) Discuss Design book design
2) Discuss who is coding / testing what in implementation plan

ASSIGNMENTS:

1) **Mike & Erich**: Finish sequence diagram updates
2) Continue to compile design book

NEXT MEETING: Thursday, 11AM, Williams 308

Meeting 13

Green Team Minutes

TIME / PLACE: *2 PM, 07 DEC 2008, Williams 308*
IN ATTENDANCE: *Erich, Matt, Todd, Mike*
FACILITATOR: *Erich*
SECRETARY: *Mike*
BODY:

1) Test Plan

2) revisions on sequence diagrams

3) back references (how will we?)

ASSIGNMENTS:

1) ALL: Do your coding and testing.

2) ALL: fix your sequence diagrams
(see "REPO://CHANGELOG_README.txt" and
"REPO://DesignDocumentation/08_SequenceDiagrams/matt_changes.txt")

3) ALL: check back through documentation and reflect any changes to your sequence diagrams in documentation

4) ALL: let the group know changes in "REPO://CHANGELOG_README.txt" file

NEXT MEETING: *TBD*

Part XV
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